RAPPORT DE STAGE

AGL pour bases de données scientifiques

Atelier de Génie Logiciel pour la conception de bases de données

Pierre Burc

Master 2 - Informatique

Spécialité : **Architectures et ingénierie du logiciel et du web (AIGLE)**

Université Montpellier II

Année 2013 – 2014

Stage effectué au CIRAD – UMR TETIS
Du 31/03/14 au 31/08/14

Directeurs de stage en entreprise
Michel Passouant,
Mathieu Roche, Sophie Fortuno

Responsable de stage de l'université
Clémentine Nébut

CIRAD – UMR TETIS

RESUME

Le projet décrit dans ce rapport s'inscrit dans le contexte du projet « Patrimoine Numérique » enclenché par le CIRAD. Au cours de ces six mois de stage, l'objectif a été d'implémenter un outil qui, à partir d'un Modèle Conceptuel de Données (MCD), était capable de fournir en sortie le Modèle Physique de Données (MPD) associé. L'étape intermédiaire consistant à établir le Modèle Relationnel de Données (MRD) était également primordiale.

Afin de répondre à cette problématique, de nombreux logiciels de modélisation UML libres ont été testés dans le but d'utiliser celui qui répondait le mieux aux besoins du projet. Une des caractéristiques essentielles pour ces logiciels était la gestion d'un format nommé XMI. C'est à l'aide de ce format que nous avons implémenté des parseurs capables de réaliser les étapes citées précédemment.

Mots-clés : Bases de données, MCD, MRD, MPD, parseur, logiciel de modélisation UML, XMI.

Remerciements

Je tiens à remercier tout particulièrement mes maîtres de stage M. Michel Passouant et M. Mathieu Roche pour m'avoir suivi de manière régulière tout au long du stage, ainsi que pour leur disponibilité et leurs précieux conseils.

Je remercie également Mme Sophie Fortuno pour avoir participé aux réunions et partagé son expérience importante dans le domaine de la gestion des données.

Je tiens également à remercier M. Guillaume Cornu pour son aide précieuse quant à la partie technique du projet.

Je remercie Mme Nathalie Saint-Geours pour m'avoir proposé de travailler sur son sujet de thèse afin d'avoir un cas d'étude concret sur lequel tester le bon fonctionnement de mes logiciels.

Enfin, je remercie l'intégralité du bâtiment F du campus de Baillarguet, bâtiment dans lequel j'ai été affecté et où j'ai été accueilli chaleureusement par l'ensemble du personnel, ainsi que pour la bonne humeur qui a régné durant ces cinq mois.

Table des matières

Table des illustrations

I. Introduction

1. Présentation du CIRAD

Le CIRAD est un centre de recherche français qui répond, avec les pays du Sud, aux enjeux internationaux de l'agriculture et du développement.

Etablissement public à caractère industriel et commercial, le CIRAD (Centre de coopération internationale en recherche agronomique pour le développement) est placé sous la double tutelle du ministère de l'Enseignement supérieur et de la Recherche et du ministère des Affaires étrangères et européennes.

En partenariat avec les pays du Sud dans leur diversité, le CIRAD produit et transmet de nouvelles connaissances, pour accompagner leur développement agricole et contribuer au débat sur les grands enjeux mondiaux de l'agronomie.

Organisme de recherche finalisée, le CIRAD établit sa programmation à partir des besoins du développement, du terrain au laboratoire, du local au planétaire. Ses activités relèvent des sciences du vivant, des sciences sociales et des sciences de l'ingénieur appliquées à l'agriculture, à l'alimentation et aux territoires ruraux. Le CIRAD s'engage au plus près des hommes et de la Terre sur des défis complexes et évolutifs : sécurité alimentaire, gestion des ressources naturelles, inégalités et lutte contre la pauvreté.

L'organisme concentre ses recherches autour de 6 axes scientifiques prioritaires. Il est présent en priorité dans le cadre de dispositifs de recherche et d'enseignement en partenariat.

Le CIRAD dispose d'un réseau mondial de partenaires et de 12 directions régionales, à partir desquelles il mène des activités de coopération avec plus de 90 pays.

En France, il met à la disposition de la communauté scientifique nationale et internationale un important dispositif de recherche et de formation situé principalement à Montpellier et dans l'outre-mer français.

Le CIRAD est membre fondateur d'Agreenium, le Consortium national pour l'agriculture, l'alimentation, la santé animale et l'environnement, et membre d'AllEnvi, l'Alliance nationale de recherche pour l'environnement.

Le CIRAD en chiffres :

- 1800 agents, dont 800 chercheurs.
- Des activités en partenariat avec plus de 90 pays.

- 3 départements scientifiques : Systèmes biologiques (Bios), Performance des systèmes de production et de transformation tropicaux (Persyst), Environnement et sociétés (ES).
- 36 unités de recherche.
- 12 directions régionales en France métropolitaine, dans l'outre-mer français et à l'étranger.
- Une trentaine d'outils collectifs de recherche accessibles aux partenaires du Sud et aux partenaires européens.
- Près de 5 millions d'euros consacrés à la formation doctorale.
- Plus de 300 doctorants encadrés chaque année, dont 60% originaires d'un pays du Sud.
- Des masters internationaux créés avec les grandes écoles ou les universités.
- 800 chercheurs et techniciens du monde entier accueillis et formés chaque année.
- Un budget annuel de 218 millions d'euros.

Le site de Montpellier

Situé au sein d'un campus mondial de recherche agricole pour le développement, le centre de Montpellier bénéficie d'un environnement scientifique exceptionnel. Le CIRAD est membre d'Agropolis Fondation et d'Agropolis International, lieu d'échanges et de débats, où les scientifiques, français et étrangers, viennent confronter leurs connaissances et leurs idées.

2. Présentation de l'unité mixte de recherche – UMR TETIS

TETIS - Territoire, Environnement, Télédétection et Information Spatiale - est une unité mixte de recherche sous la tutelle des institutions AgroParisTech, CIRAD et Irstea.

Ces trois institutions ont confié à l'UMR TETIS la mission de développer la maîtrise de l'information spatiale pour la recherche agro-environnementale et pour la gestion durable des territoires, en mettant à disposition des agents de chaque institution.

L'unité mène des recherches sur les méthodes d'analyse et de représentation spatiale de systèmes agroenvironnementaux et territoriaux, selon quatre axes : l'analyse des structures et des dynamiques spatiotemporelles de ces systèmes et de leur perception par les acteurs ; l'acquisition et le traitement des données spatialisées (télédétection, MNT, GPS…) ; l'ingénierie des systèmes d'information ; les démarches de développement territorial et de gestion agroenvironnementale ainsi que la place des systèmes d'information.

C'est dans cette unité mixte de recherche que se trouvent mes trois maîtres de stage, M. Michel Passouant, M. Mathieu Roche et Mme Sophie Fortuno. M. Passouant est statisticien informaticien, spécialiste dans le traitement d'enquêtes et en conception de systèmes d'information. M. Roche est un chercheur du CIRAD en informatique, spécialisé

notamment dans les systèmes d'informations et la fouille de textes. Mme Fortuno est chef de projet spécialisée dans les systèmes d'informations.

3. Problématique du stage

Depuis quelque temps, le CIRAD est engagé dans une démarche de réflexion sur la nature et la pérennisation de son patrimoine numérique. Cette réflexion met en évidence trois objectifs à atteindre.

Le premier d'entre eux est de faire un inventaire très précis de l'ensemble des données déjà produites. Ceci nécessite une entière collaboration de l'ensemble du personnel du CIRAD car il faut que chacun signale les données en sa possession, que ce soient des données issues de travaux de recherche ou bien des données plus générales.

Le deuxième objectif est de stocker et organiser l'ensemble de ces données dans un lieu accessible par l'ensemble du personnel du CIRAD, afin de pouvoir en découvrir l'existence et les consulter.

Enfin, le dernier objectif est de permettre la réutilisation de ces données pour de futurs travaux de recherche ou autres. Pour cela, il faut impérativement qu'elles soient documentées à l'aide de métadonnées, car dans le cas contraire, les données sont très souvent voire systématiquement inexploitables. Ce projet permettra également de conserver les données en cours de production ainsi que celles créées dans un avenir plus ou moins proche.

Ce projet requiert un très long processus pour atteindre l'ensemble des objectifs fixés, spécialement pour les travaux de recherche en cours ou déjà terminés. Pour chacun d'entre eux, les mettre à disposition de tous ne sera pas un réel problème, mais le plus compliqué sera d'avoir le contexte exact dans lequel les données récupérées et calculées ont été obtenues, et de vérifier également la structuration de celles-ci, la façon dont elles sont gérées.

Mon stage s'inscrit dans le cadre de ce projet, avec pour objectif principal d'accompagner les chercheurs dans la construction des jeux de données pour leurs futurs travaux de recherches. Le but est donc de mettre l'accent sur la première phase, étant celle de la conception et de modélisation de leurs données. De cette manière, en partant dès le départ sur une structure cohérente avec des données documentées et liées entre elles, valider le processus évoqué précédemment sera beaucoup plus simple. Pour répondre à ce besoin, il me fallait être capable de proposer aux chercheurs une procédure stricte qui débutera par la phase de conception et terminera par la réalisation du Modèle Physique de Données (MPD) permettant l'implémentation de la base de données dans un Système de Gestion de Bases de Données (SGBD).

A cet effet, mon stage s'est déroulé au département ES (Environnement et Sociétés), dans l'unité mixte de recherche UMR TETIS. Durant les cinq mois de mon stage, j'ai été

accompagné et conseillé par M. Michel Passouant et M. Mathieu Roche. Nous avions organisé des réunions hebdomadaires pour que je puisse faire un bilan de mon état d'avancement ainsi que pour prendre les décisions adéquates ensemble. Au fur et à mesure de l'avancée du projet, d'autres personnes ont participé activement aux réunions, comme Mme Sophie Fortuno et M. Guillaume Cornu.

Afin de visualiser clairement les différents objectifs qui m'ont été donnés, la rédaction d'un cahier des charges était nécessaire.

4. Cahier des charges

a) Les objectifs

A ce stade, il peut être légitime de penser qu'un tel exercice n'est pas nécessaire car il existe déjà sur le marché des ateliers de génie logiciel (WinDesign ou PowerAMC pour ne citer qu'eux) qui accompagnent de A à Z les personnes désirant faire ce qui a été résumé précédemment. Seulement, ces différents AGL souffrent d'un certain nombre de défauts, tels que :

- La complexité de leur utilisation liée à la richesse des fonctionnalités proposées (la plupart inutiles pour ce que veut en faire le CIRAD), donc très long à prendre en main.
- Un coût très important pour se les procurer, du fait des nombreuses fonctionnalités offertes.
- La dimension spatio-temporelle trop peu souvent gérée, et pourtant essentielle pour la mise en place de base de données pour le CIRAD.
- Une gestion des métadonnées insuffisante pour répondre aux exigences du projet patrimoine numérique.

En raison de ces points cités, mes tuteurs avaient fixé comme contrainte d'utiliser un logiciel libre fondé sur le formalisme UML. La force de ces programmes est évidemment leur coût inexistant mais également leur utilisation plus intuitive et plus rapide du fait de leurs fonctionnalités réduites à l'essentiel. Pour rappel, un logiciel est dit « libre » s'il respecte les points cités ci-dessous :

- ✓ La liberté d'exécuter le programme pour tous les usagers
- ✓ La liberté d'étudier le fonctionnement du programme et de l'adapter à ses besoins
- ✓ La liberté de redistribuer des copies du programme
- ✓ La liberté d'améliorer le programme et de distribuer ces améliorations au public

Ces libertés impliquent la possibilité d'avoir accès au code source. Le plus souvent, les logiciels libres sont sous Licence Public Générale (GPL), indiquant que tous les composants utilisés ou créés pour le programme sont également sous cette licence.

Cependant, en l'état, un tel logiciel ne répond pas à toutes les attentes fixées par mes tuteurs, car ces derniers permettent uniquement de réaliser des Modèles Conceptuels de Données.

Par convention, pour la suite du rapport, voici le récapitulatif du vocabulaire utilisé concernant les différents modèles.

Diagramme de classes / Modèle Conceptuel de Données	Modèle Relationnel de Données	Modèle Physique de Données
Classe / Entité	Relation / Table	Table
Attribut	Attribut	Champ
Identifiant	Identifiant / Clé primaire	Clé primaire
✘	Clé étrangère	Clé étrangère
Association	Association	Lien
Multiplicité / Cardinalité	Multiplicité	✘

Figure 1 : Vocabulaire conventionnel utilisé

Mon rôle est donc de pallier les limites du logiciel choisi à l'issue d'une étude approfondie en l'étendant à l'aide des fonctionnalités nécessaires, dont notamment deux parseurs :

- Le premier transformant le Modèle Conceptuel de Données (MCD) en Modèle Relationnel de Données (MRD).
- Le second, transformant le Modèle Relationnel de Données en script SQL.

Un parseur est un programme qui permet d'analyser un fichier et d'en déterminer sa structure syntaxique afin d'effectuer divers traitements, comme par exemple des transformations dans ce projet.

Comme dit précédemment, il fallait également prendre en compte la dimension spatio-temporelle, du fait que les bases de données conçues par le CIRAD sont souvent axées sur le secteur agricole et rural (enquêtes et gestion d'exploitations, planification et gestion de territoires…) et que nativement, aucun programme libre ne gère ces types de données (point, ligne, polygone).

- Intégrer et gérer la notion de données spatio-temporelles, avec les types tels que Point, Ligne et Polygone.

L'implémentation des deux parseurs a nécessité un travail en amont consistant à analyser de manière très approfondie le langage XMI, expliqué dans la partie suivante. Concernant le premier parseur permettant de passer du MCD au MRD, les règles de transformation standards devaient être appliquées, et cela n'était pas simple car il fallait envisager tous les cas de figures possibles, comprenant un bon nombre de cas spéciaux à gérer. Celles-ci seront listées plus tard dans le rapport.

Le but recherché à travers ce projet est de simplifier toute la phase de préparation de la base de données pour les chercheurs. En effet, seule l'étape de modélisation devra être

effectuée, le reste étant entièrement automatisé. De cette manière, ces derniers pourront se focaliser sur cette phase de réflexion primordiale pour la suite de leurs travaux, afin de jeter les fondements de bases de données cohérentes et normalisées. D'autre part, ce projet pourra servir de formation dans les pays du sud, du fait que les compétences liées à l'informatique et à des méthodologies en rapport avec UML sont moins développées.

La figure ci-dessous illustre les différentes étapes ainsi que le cheminement du projet.

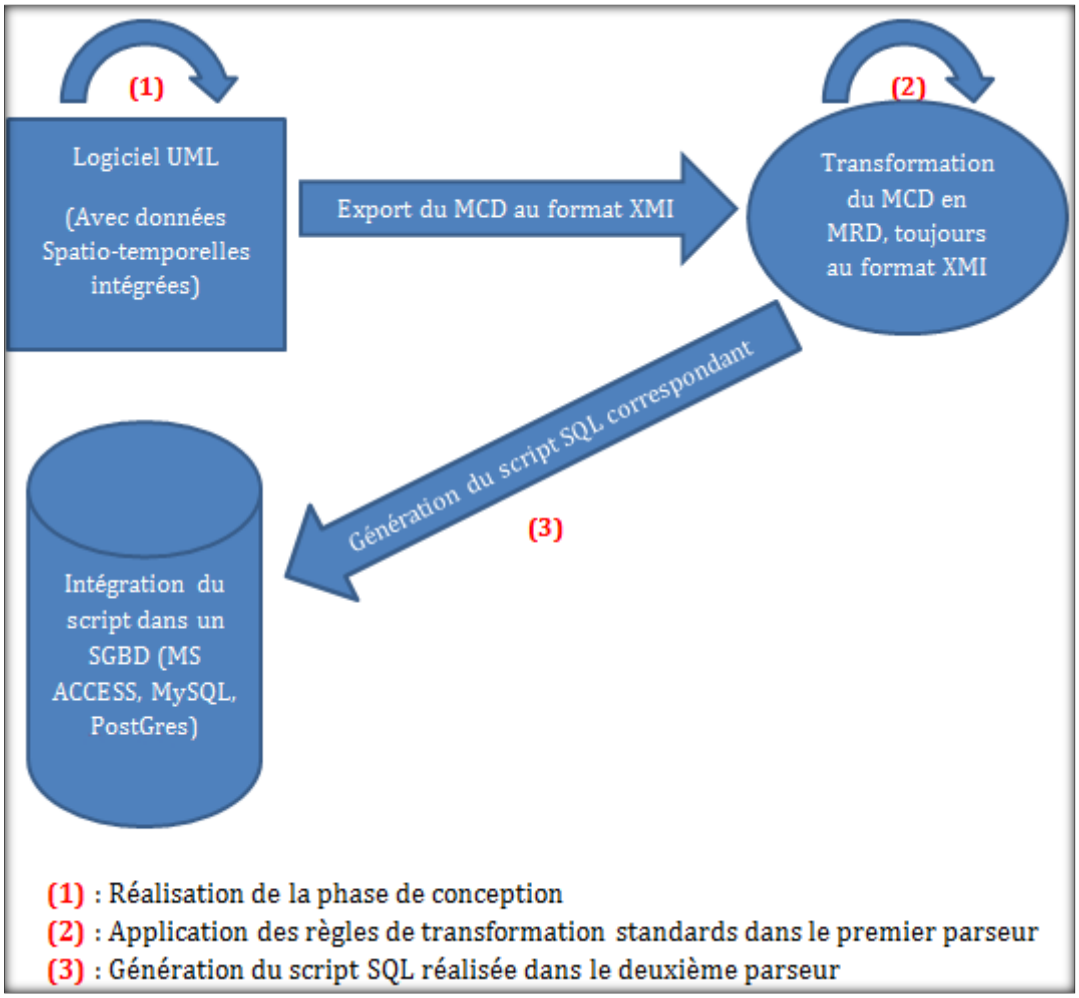

Figure 2 : Etapes composant le projet

b) Les livrables attendus

A l'issue de ce stage, plusieurs livrables étaient demandés par mes tuteurs, comprenant :

- Le logiciel libre choisi, avec la gestion des données spatio-temporelles
- L'implémentation du premier parseur utilisable via une interface graphique
- L'implémentation du second parseur utilisable via une interface graphique

- Un manuel utilisateur guidant pas à pas les différentes étapes à réaliser (fourni dans l'annexe technique)
- Le présent rapport

Pour faciliter la distribution, les deux parseurs ainsi que le manuel utilisateur et d'autres fichiers annexes devaient être fournis via un installeur, donc par un programme exécutable (.exe).

Maintenant que le contexte ainsi que les différents objectifs sont posés, une analyse des différents logiciels de modélisation UML libres présents sur le marché s'impose.

II. Etat de l'art

Afin de débuter un projet dans de bonnes conditions, il est nécessaire de réaliser une analyse de l'existant. Dans notre cas, nous avons dû comparer les logiciels UML Open Source existant sous Windows. Le but était de dégager les spécificités de chacun d'entre eux, et plus particulièrement la façon dont ils géraient les diagrammes de classes, dans le cadre de la réalisation de Modèle Conceptuel de Données.

L'élément central et récurrent de cette analyse a été la manière dont le MCD était représenté lorsqu'on l'exportait. En effet, à quelques différences près, l'ensemble des logiciels UML procurent les mêmes fonctionnalités pour créer un MCD. Cependant, la façon dont celui-ci était géré par la suite différait plus ou moins selon ces logiciels.

Nous nous sommes donc concentrés sur ceux qui proposaient l'export au format XMI, ce qui m'a notamment permis d'éliminer d'entrée Open ModelSphere.

1. Le format XMI

Le XMI (XML Metadata Interchange) est un format d'échange standard entre ateliers de génie logiciel, fondé sur le langage XML. Ce format permet de représenter un Modèle Conceptuel de Données dans un langage balisé. Par exemple, si un diagramme de classes est réalisé à l'aide du logiciel ArgoUML et qu'il est exporté au format XMI, il peut théoriquement être importé dans un autre logiciel UML, tel que BOUML. Malheureusement, cela n'est pas si simple car il existe une multitude de versions du format XMI. D'une version à l'autre, nous pouvons noter :

- La création ou le changement de nom d'une balise
- Le nombre d'attributs pour une balise (Nouveaux attributs ou suppression)

Ces différences entraînent des structures de fichier très variables selon la version XMI utilisée. De plus, ce format est associé à deux autres standards qui sont UML et MOF, possédant également de multiples versions, rendant encore plus complexe l'échange.

Cependant, ce format ne nous intéresse pas pour sa fonction première qui est l'export / import. En effet, notre objectif n'est pas de réaliser un diagramme de classes pour l'ouvrir dans plusieurs logiciels de modélisation UML différents, mais de transformer ce diagramme de classes en Modèle Relationnel de Données. Or, passer par le format XMI est tout indiqué car l'utilisation d'un langage balisé pour représenter un MCD va être très utile pour réaliser un parseur et pouvoir effectivement passer à un MRD.

La structure des différents formats XMI générés par les différents logiciels sera donc un paramètre essentiel à prendre en compte. La version XMI utilisée à l'export sera tout

de même un argument de poids, car plus celle-ci est récente, plus la structure générée est claire et relativement simple à analyser.

Afin de bien comprendre la notion de format XMI, la première figure ci-dessous représente un diagramme de classes réalisé sous BOUML, et la seconde le même diagramme de classes exporté au format XMI.

Figure 3 : Diagramme de classes réalisé sous BOUML

```
<packagedElement xmi:type="uml:Package" xmi:id="BOUML_0x81_22" name ="Exemple_Rapport">
    <packagedElement xmi:type="uml:Class" name="ClasseA" xmi:id="BOUML_0x1f484_4" visibility="package">
        <ownedAttribute xmi:type="uml:Property" name="" xmi:id="BOUML_0x1f484_0" association="BOUML_ASSOC_0x1f484_0" aggregation="none">
            <type xmi:type="uml:Class" xmi:idref="BOUML_0x1f504_4"/>
            <lowerValue xmi:type="uml:LiteralString" xmi:id="BOUML_MULTIPLICITY_L_0x1f484_0" value="*"/>
            <upperValue xmi:type="uml:LiteralString" xmi:id="BOUML_MULTIPLICITY_U_0x1f484_0" value="*"/>
        </ownedAttribute>
        <ownedAttribute xmi:type="uml:Property" name="AttributA" xmi:id="BOUML_0x1f484_1" visibility="private">
        </ownedAttribute>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" name="ClasseB" xmi:id="BOUML_0x1f504_4" visibility="package">
        <ownedAttribute xmi:type="uml:Property" name="" xmi:id="BOUML_0x1f504_0" association="BOUML_ASSOC_0x1f484_0" aggregation="none">
            <type xmi:type="uml:Class" xmi:idref="BOUML_0x1f484_4"/>
            <lowerValue xmi:type="uml:LiteralString" xmi:id="BOUML_MULTIPLICITY_L_0x1f504_0" value="1"/>
            <upperValue xmi:type="uml:LiteralString" xmi:id="BOUML_MULTIPLICITY_U_0x1f504_0" value="1"/>
        </ownedAttribute>
        <ownedAttribute xmi:type="uml:Property" name="AttributB" xmi:id="BOUML_0x1f504_1" visibility="private">
        </ownedAttribute>
    </packagedElement>
    <packagedElement xmi:type="uml:Association" xmi:id="BOUML_ASSOC_0x1f484_0" name="AssociationAB" visibility="private">
        <memberEnd xmi:idref="BOUML_0x1f484_0"/>
        <memberEnd xmi:idref="BOUML_0x1f504_0"/>
    </packagedElement>
</packagedElement>
```

Informations relatives à une classe
Informations relatives à un attribut présent dans une classe
Informations relatives à une association
Informations relatives aux multiplicités de l'association

Figure 4 : XMI correspondant au diagramme de classes

2. Analyse des logiciels UML

En plus de la gestion du format XMI, l'analyse des logiciels s'est portée également sur la simplicité d'utilisation et les différents éléments d'un diagramme de classes proposés pour la conception de modèles.

a) ArgoUML

Le premier logiciel testé était ArgoUML. Celui-ci est implémenté en JAVA et sa dernière version stable date du 15 décembre 2011.

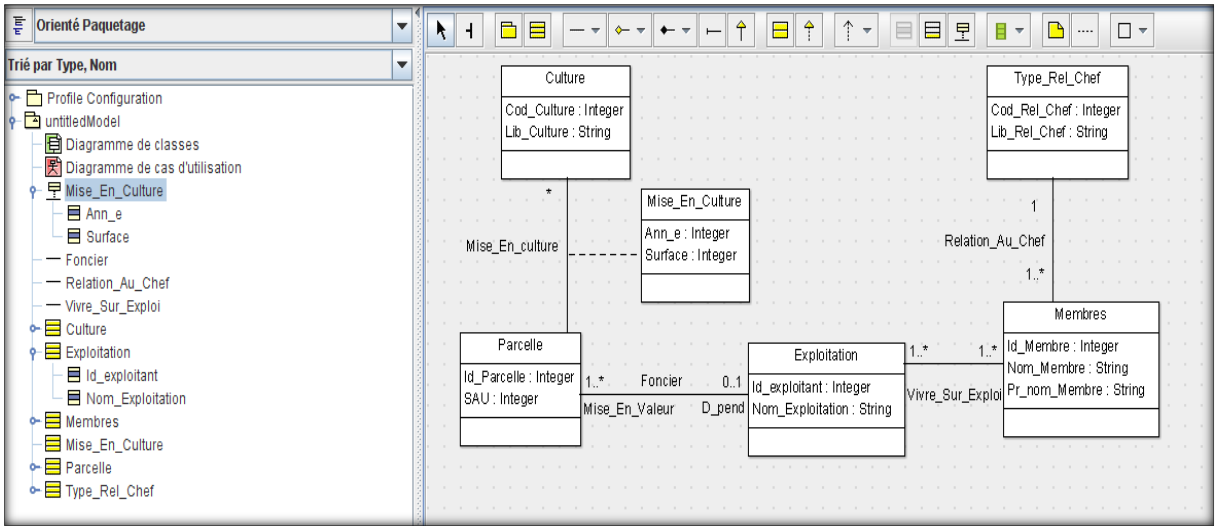

Figure 5 : Interface de ArgoUML

Comme le montre la figure 5, l'interface proposée est assez intuitive, nous permettant de réaliser rapidement un MCD en intégrant des classes, classes d'associations, lien simple, de composition, d'agrégation et de spécialisation / généralisation, ainsi que des énumérations. L'ajout d'un élément se fait par un simple cliquer-glisser, comme l'ensemble des logiciels présentés par la suite. Concernant le XMI, ArgoUML gère l'export au format XMI 1.2 associé au standard UML 1.4. Il faut savoir que la version actuelle de XMI est la 2.4.2 datant d'avril 2014 et la version UML 2.4.1 d'août 2011, mais l'ensemble des informations essentielles seront récapitulées dans un tableau comparatif à la fin des différentes analyses. ArgoUML utilise donc des standards assez anciens, qui ne sont pas forcément compatibles avec les versions les plus récentes, comme le montre la figure 6.

	XMI1.0 DTD	XMI1.1 DTD	XMI1.2 DTD	XMI1.2 Schema	XMI2.0 Schema	XMI2.1 Schema
UML1.3	X	X	X	X		
UML1.4	X	X	X	X	X	
UML1.5	X	X	X	X	X	
UML2.0						X

Figure 6 : Compatibilité versions XMI et UML

ArgoUML n'est donc pas capable d'interagir avec les versions les plus récentes, car outre l'export, il n'importe que des versions XMI 1.1 et 1.2 avec UML 1.4.

b) BOUML

BOUML est quant à lui un logiciel écrit en C++ sous Qt3, sa dernière version stable Open Source datant du 1er octobre 2011.

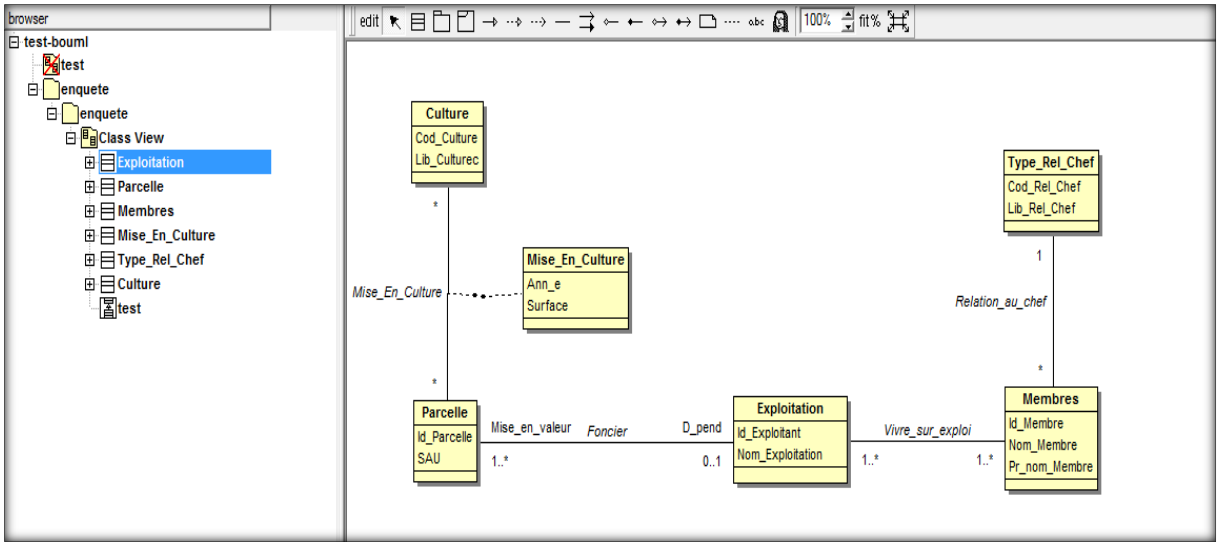

Figure 7 : Interface de BOUML

Tout comme ArgoUML, BOUML propose une interface simple et épurée permettant de modéliser les principaux éléments d'un MCD. J'ai d'ailleurs trouvé son utilisation encore plus intuitive et plus permissive, car ArgoUML présentait quelques défauts et quelques bugs pour faire apparaître certains éléments, notamment manipuler les classes d'association. Ici, le logiciel propose plusieurs versions XMI à l'export et à l'import, indiquées en figure 8.

Compatibilité	XMI 1.2 (Export)	XMI 2.0 (Import / Export)	XMI 2.1 (Import / Export)
UML 1.4	✓		
UML 2.0		✓	
UML 2.1			✓
UML 2.2			✓
UML 2.3			✓

Figure 8 : Versions XMI et UML d'import / export pour BOUML

Comme l'indique ce tableau, BOUML utilise des versions beaucoup plus récentes concernant XMI et UML et offre un panel plus large. Le seul reproche pouvant être noté pour BOUML est l'impossibilité de réaliser des associations n-aires, bien que peu utilisées dans des cas concrets.

c) Modelio

Modelio est un outil de modélisation UML implémenté en JAVA sous Eclipse, sa dernière version datant du 24 février 2014.

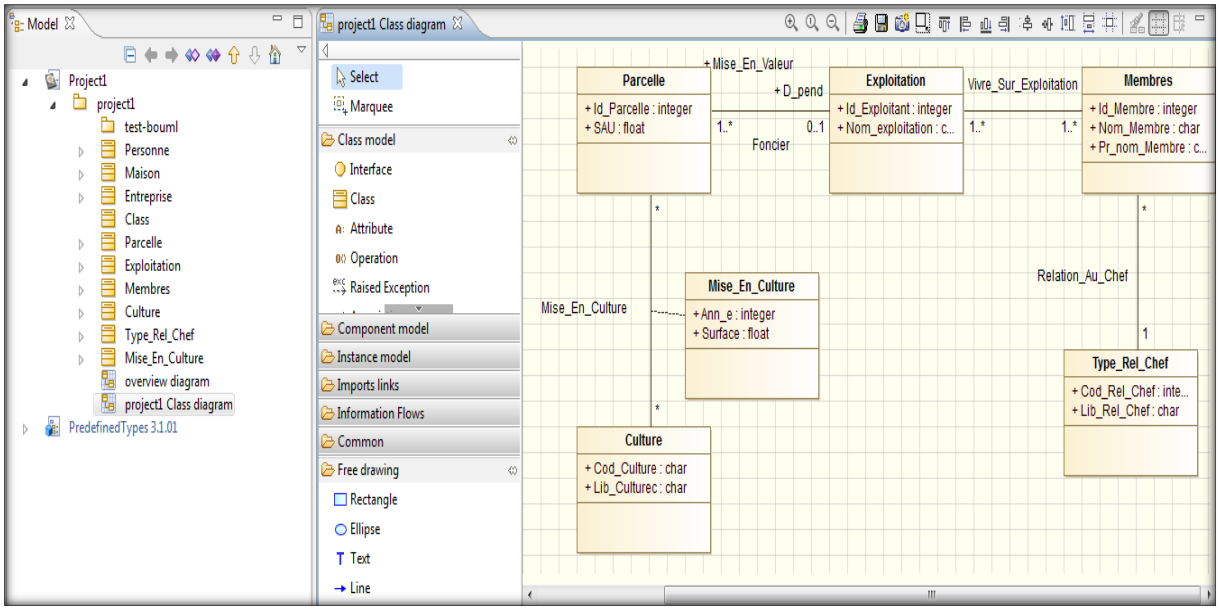

Figure 9 : Interface de Modelio

L'interface proposée par Modelio est un peu plus complexe à prendre en main, mais propose les mêmes fonctionnalités. Seul le XMI 2.1 est supporté par cet outil, avec les mêmes versions UML que pour BOUML, avec en plus le dernier standard UML 2.4.1. Bien que la version utilisée de XMI soit identique à celle de BOUML, la comparaison détaillée des deux fichiers générés comportent des différences dans la structure globale, et il était plus difficile pour moi de cerner la logique instaurée par Modelio.

Le gros point négatif de Modelio est sa lourdeur du fait qu'il embarque sa propre version d'Eclipse et la façon dont gérer les différents éléments constituant un diagramme de classes.

d) StarUML

StarUML est un logiciel implémenté en Delphi, sa dernière version datant du 30 décembre 2005. L'interface ci-dessous montre encore une fois la simplicité d'utilisation de l'outil. Comme ArgoUML, le format XMI supporté est ancien, car seul le XMI 1.1 couplé à UML 1.3 est proposé à l'export.

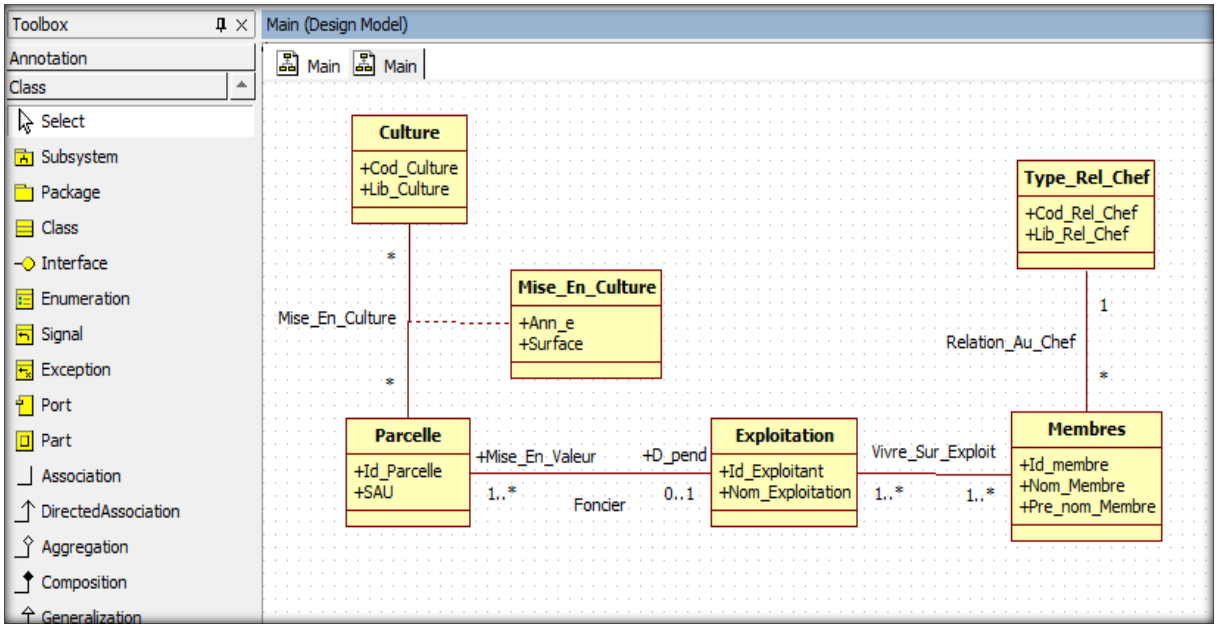

Figure 10 : Interface de StarUML

D'autre part, sa version commence à dater quelque peu (2005), et sa communauté n'est plus beaucoup active.

3. Comparaison des logiciels UML

Voici donc les principaux logiciels de modélisation UML Open Source testés en début de stage. Le tableau ci-dessous recense l'ensemble des informations citées précédemment.

	ArgoUML	BOUML	Modelio	StarUML
Dernière version stable	15/12/11	1/10/11	24/02/14	30/12/05
Langage de programmation utilisé	JAVA	C++	JAVA	Delphi
Couples XMI/UML à l'export	XMI 1.2 / UML 1.4	XMI 1.2 / UML 1.4 XMI 2.0 / UML 2.0 XMI 2.1 / UML 2.1 XMI 2.1 / UML 2.2 XMI 2.1 / UML 2.3	XMI 2.1 / UML 2.1.1 XMI 2.1 / UML 2.2 XMI 2.1 / UML 2.3 XMI 2.1 / UML 2.4.1 XMI 2.1 / EMF UML 3.0.0	XMI 1.1 / UML 1.3
Couples XMI/UML à l'import	XMI 1.1/ UML 1.4 XMI 1.2 / UML 1.4	XMI 2.0 / UML 2.0 XMI 2.1 / UML 2.1 XMI 2.1 / UML 2.2 XMI 2.1 / UML 2.3	XMI 2.1 / UML 2.1 XMI 2.1 / UML 2.2 XMI 2.1 / UML 2.3	XMI 1.1 / UML 1.3
Version actuelle de XMI	Avril 2014 : 2.4.2			
Version actuelle de UML	Août 2011 : 2.4.1			
Remarques	Outil complet pour la modélisation mais trop limité pour la partie XMI.	Bénéficie d'une communauté active et dispose d'un manuel d'utilisation clair.	Comporte des modules payant.	Peu utilisé actuellement, versions XMI vieilles.

Figure 11 : Récapitulatif des différents logiciels UML

Dans un premier temps, mes choix se sont donc portés vers BOUML ainsi que vers Modelio, offrant chacun de nombreuses possibilités pour gérer l'export de fichiers XMI, avec des versions récentes.

Pour la partie modélisation, l'interface proposée par BOUML est plus intuitive et plus accessible à des utilisateurs qui ne sont pas obligatoirement informaticiens. D'autre part, il existe une documentation assez claire et BOUML bénéficie d'une communauté non négligeable. De son côté, Modelio est aussi complet que BOUML dans les

fonctionnalités offertes pour réaliser un diagramme de classes. Cependant, comme nous l'avons dit précédemment, sa prise en main est un peu plus difficile pour des personnes non-initiées.

Ensuite, le XMI est géré à peu près de la même façon, à la différence près que BOUML gère aussi bien le XMI 1.X que 2.X, tandis que Modelio se concentre uniquement sur le XMI 2.1, en proposant le standard UML actuel.

J'ai donc analysé en détail le corps des fichiers générés par l'un et par l'autre, et bien que les versions utilisées soient identiques, des différences sont visibles dans la construction du fichier XMI. Principalement, la manière dont les deux logiciels gèrent les descriptions des associations est intéressante.

Afin de comparer ces deux logiciels, basons-nous à nouveau sur le diagramme de classes basique déjà présenté précédemment. A titre d'exemple, voici sa représentation sous Modelio.

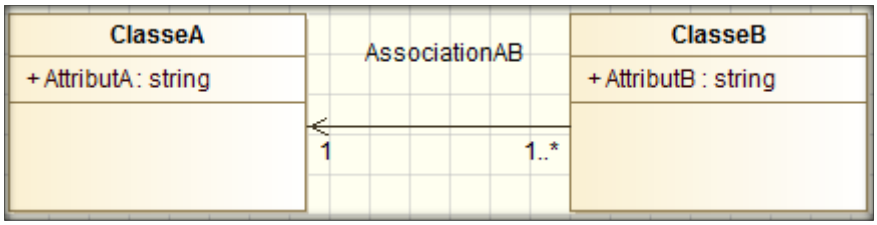

Figure 12 : Diagramme de classes réalisé sous Modelio

a) Analyse détaillée du format XMI de BOUML

Le format XMI analysé de BOUML est la version 2.1 couplée avec la version 2.3 d'UML. Reprenons la figure déjà présentée pour expliquer en détail la structure du fichier, et plus spécialement les informations relatives à « ClasseA ».

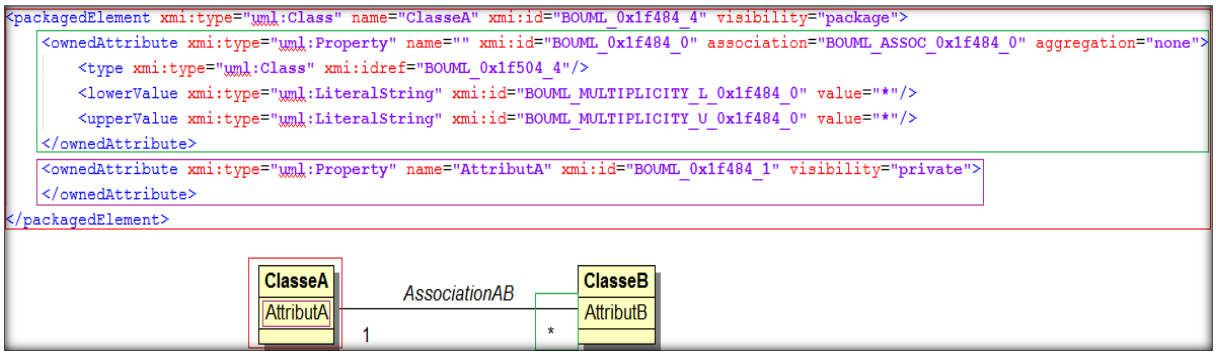

Figure 13 : Analyse du format XMI de BOUML

Les cadres colorés permettent d'établir la correspondance entre le diagramme de classes présenté de manière graphique avec son format XMI. Le XMI de BOUML est relativement rapide à prendre en main et à comprendre. En effet, il est assez intuitif car structuré de manière cohérente.

La première balise (Cadres rouge) nommée « PackagedElement » représente « ClasseA » dans son intégralité. Nous retrouvons dans ses attributs le nom de la classe grâce à « name », son type, ici « uml :Class » et son identifiant sous « xmi :id ». L'identifiant joue un rôle très important car chaque élément possède son identifiant unique. De cette manière, il est simple de faire référence à un élément précis, en faisant référence à son identifiant.

La seconde balise (Cadres violet) imbriquée nommée « ownedAttribute » décrit les informations relatives à l'attribut « AttributA ». Comme pour la classe, nous pouvons retrouver pour ce dernier son nom, son identifiant et son type. Bien entendu, cette balise apparaîtra autant de fois qu'il y a d'attributs dans la classe.

Enfin, la première balise imbriquée (Cadres vert) nommée également « ownedAttribute » contient les informations relatives à l'extrémité droite de l'association « AssociationAB ». Les mêmes informations sont ici présentes, avec deux attributs supplémentaires qui sont « association », faisant référence à l'identifiant de « AssociationAB » présentée juste après, et « aggregation », spécifiant si ce côté de l'association correspond à une composition, une agrégation ou un simple lien d'association. La valeur « none » indique qu'il s'agit de la dernière proposition citée. C'est grâce à la présence de ses deux attributs supplémentaires que nous pouvons savoir si la balise « ownedAttribute » décrit un attribut ou une extrémité d'association. De plus, cette balise en contient trois autres, dont :

- **type** : son attribut « xmi :idref » permet de savoir quelle classe se trouve de l'autre côté de l'association (ici xmi :idref contient l'identifiant de « ClasseB »).
- **lowerValue** : comme son nom l'indique, précise qu'elle est la valeur de la multiplicité minimale. Ici, ayant choisi « * » dans BOUML, lowerValue vaut « * ».
- **upperValue** : précise la valeur maximale de la multiplicité, ici « * ».

A travers cette analyse, nous nous rendons compte qu'il serait même possible de reconstruire manuellement un diagramme de classes uniquement avec en notre possession son format XMI, même si bien sur cette opération serait assez longue et laborieuse mais pas impossible.

Il est inutile de présenter le format XMI de « ClasseB » car sa structure sera identique à celle de « ClasseA ». Cependant, il est intéressant de voir la structure représentant « AssociationAB » dans son intégralité.

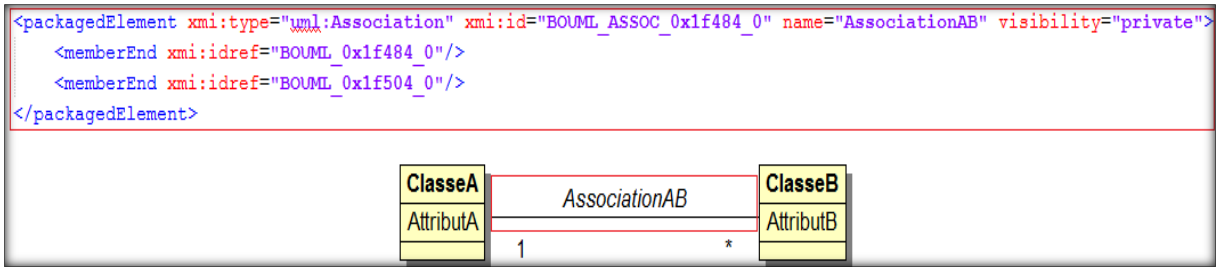

Figure 14 : Suite de l'analyse du format XMI de BOUML

Tout comme une classe, la première balise se nomme « packagedElement ». Pour distinguer une balise représentant une classe d'une balise représentant une association, il suffit de regarder la valeur de l'attribut « xmi :type », car ici sa valeur est « uml :Association » tandis que précédemment cette dernière était « uml :Class ».

C'est ici que nous retrouvons le nom de l'association, ainsi que son identifiant. Pour établir le lien entre cette structure et celle présente dans une classe sous la balise « ownedAttribute », il y a un système de double référencement, comme le montre la figure ci-dessous.

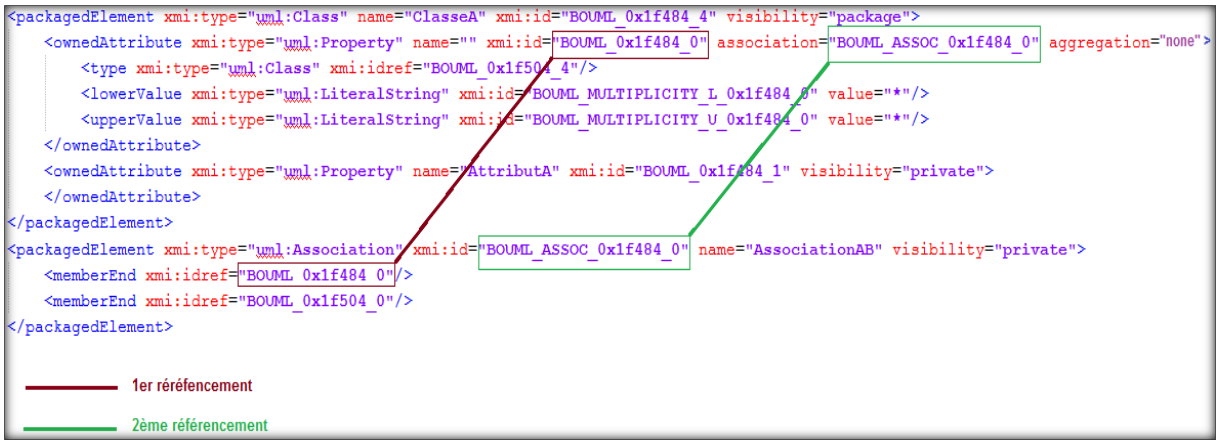

Figure 15 : Double référencement XMI BOUML

Vous l'aurez compris, les deux balises imbriquées nommées « memberEnd » font référence aux deux extrémités de l'association « AssociationAB », l'une présente dans « ClasseA » et l'autre dans « ClasseB ».

Il est donc facile de naviguer à travers le XMI en utilisant le système de référencement. L'avantage de BOUML est qu'il génère un fichier avec uniquement les informations dont nous avons besoin pour reconstituer le diagramme de classes, il n'ajoute pas des balises inutiles qui pourraient être susceptibles d'interférer dans notre lecture du fichier et donc dans notre compréhension.

b) Analyse détaillée du format XMI de Modelio

Voici maintenant l'analyse du format XMI fourni par Modelio en s'appuyant sur la figure 16. A préciser que la version choisie ici est identique à BOUML, c'est-à-dire la version 2.3 de XMI.

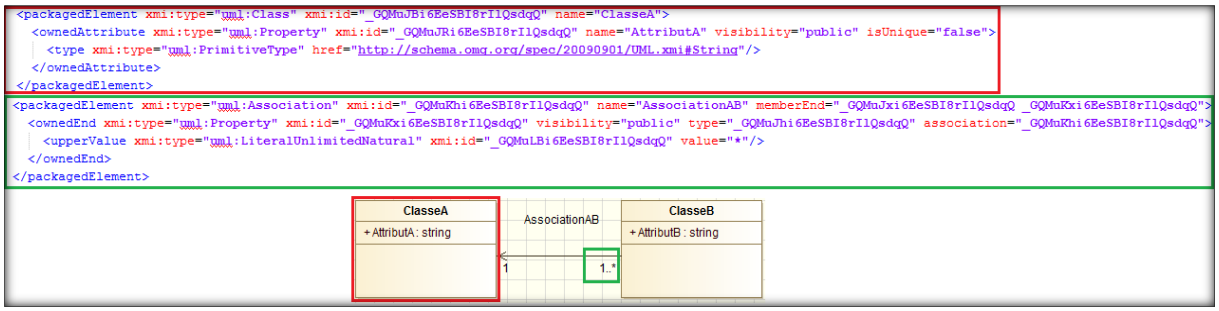

Figure 16 : Analyse du format XMI de Modelio

La première balise « packagedElement » correspond à la description de « ClasseA », comme l'indique les cadres rouges. Nous retrouvons ici les mêmes attributs déjà présentés pour BOUML. Cependant, nous nous apercevons qu'il n'y a plus de balise « ownedAttribute » servant à définir l'extrémité droite de « AssociationAB ». La description de « ClasseA » ne nous permet donc pas de savoir qu'elle possède une association la reliant à « classeB ».

La seconde balise « packagedElement » correspond à l'élément « AssociationAB ». Dans celle-ci nous retrouvons uniquement une balise imbriquée nommée « ownedEnd » dans laquelle seule la multiplicité maximale est indiquée, dans la balise « upperValue ». C'est donc la retranscription des associations dans le format XMI qui rend impossible la lecture du fichier et donc ne permet d'établir un cheminement logique entre tous les éléments. Contrairement au fichier XMI de BOUML, il est ici inutile d'essayer de reconstruire un diagramme de classes en partant de son format XMI.

A la génération, nous perdons des informations cruciales, comme par exemple la multiplicité minimale pour l'extrémité de l'association. D'autre part, l'autre extrémité de l'association est retranscrite pour on ne sait quelle raison dans la balise « packagedElement » de « ClasseB », avec également perte de l'information pour la multiplicité minimale. Malgré plusieurs tests en réalisant des diagrammes de classes différents et en modifiant certaines options dont les versions de XMI, il m'a été impossible d'avoir un format XMI aussi clair et simple que celui de BOUML.

4. Synthèse

Au vu des différentes analyses et plus spécialement de la compréhension du format XMI fourni en sortie, le logiciel BOUML était celui qui convenait le mieux à notre problématique :

- Un format XMI d'une grande clarté, sans perte d'informations

- Les principaux éléments de modélisation disponibles (Classe, classe d'association, liens simples, agrégations, compositions, etc…)
- Interface intuitive, rapide à prendre en main, sans bug et épurée
- Fichiers de configuration très simples à modifier et à importer

Concernant le dernier point cité, nous reviendrons dessus plus tard dans le rapport.

III. Environnement de travail

Maintenant que l'analyse de l'existant a été réalisée et que le logiciel de modélisation UML a été choisi, nous avons pu passer à l'étape suivante consistant à choisir les différents outils pour l'implémentation de nos programmes.

1. Logiciels

a) Eclipse IDE

Eclipse est un environnement de développement (IDE) libre, extensible, universel et polyvalent. Ceci permet donc au programmeur de disposer de différents outils et d'une interface pour développer plus facilement et rapidement des applications.

Le choix s'est porté sur Eclipse pour plusieurs raisons :

- Tout d'abord, c'est un outil que j'ai utilisé durant mon cursus, je suis donc familiarisé avec son interface et ses fonctionnalités et sûr de sa fiabilité.
- Il est libre, gratuit et dispose d'un plugin Java.
- Il permet de générer une Javadoc d'une très bonne qualité aisément.
- Il permet aussi la gestion de projet, paquetage et l'importation de librairies supplémentaires en quelques clics.
- Il permet l'auto-complétion, la coloration et correction syntaxique du code en direct, de compiler et d'exécuter plus simplement que dans un terminal et d'accéder plus rapidement aux erreurs dans le code à l'aide de lien dans la console. Il permet aussi de générer des codes génériques tels que le « try/catch » pour la gestion des exceptions ou les importations de librairies.

b) Notepad++

Il me semblait important d'indiquer ce logiciel bien qu'il ne m'ait uniquement servi à lire les fichiers au format XMI. Il m'a été utile durant l'intégralité de mon stage, et c'est sous Notepad que j'ai analysé en détail la structure des fichiers, car ce dernier proposait une coloration syntaxique pour le XML très claire et lisible.

c) Oxygen

Bien que payant, j'ai pu utiliser la version d'essai de ce logiciel pour créer un XML Schema Definition (XSD) dont j'expliquerai le rôle plus tard. L'avantage d'Oxygen est qu'il possède une coloration syntaxique spéciale

conçue pour les XSD, il propose une auto-complétion, détecte et indique à la volée les erreurs de structuration du fichier.

d) SGBDs

L'objectif final du stage étant d'avoir à disposition des scripts SQL intégrables directement dans différents SGBD, j'en ai naturellement utilisé plusieurs afin de tester les scripts générés et vérifier qu'aucune erreur n'était levée lors de leur importation.

i. PGAdmin III

PGAdmin est un outil d'administration graphique pour PostgreSQL. Il est également possible d'intégrer un plugin gérant l'extension PostGIS, incontournable pour mon stage afin de tester la création de base de données possédant des données spatio-temporelles (types point, ligne ou polygone). Il fonctionne indifféremment sous un environnement Windows ou Linux.

ii. PhpMyAdmin

PhpMyAdmin (PMA) est une application Web de gestion pour les systèmes de gestion de base de données MySQL réalisée en PHP. Comme PGAdmin, PMA fonctionne sous différents environnements tels que Windows et Linux.

iii. MS ACCESS

Microsoft Access (officiellement Microsoft Office Access) est un SGBD relationnel édité par Microsoft. Il est conçu pour fonctionner sous un environnement Windows.

iv. PowerAMC

PowerAMC est un logiciel de conception créé par la société SDP qui permet de modéliser les traitements informatiques et leurs bases de données associées. Comme dit précédemment, il contient l'ensemble des fonctionnalités que je devais implémenter lors de mon stage. J'en ai donc profité pour télécharger la version d'essai afin d'étudier son comportement pour transformer le MCD en MRD, dans le cas où des transformations à effectuer me semblant complexes et peu évidentes étaient présentes. D'autre part, j'ai pu examiner les fichiers générés pour différents SGBDs, notamment MySQL et PostGreSQL, afin de voir leur spécificité et la syntaxe adoptée.

e) QGis

QGis est un logiciel de système d'information géographique (SIG) libre et multiplateformes. Il est donc capable de gérer et d'interpréter des données spatio-temporelles présentes dans des bases de données. De plus, il est très simple et rapide d'établir une connexion entre QGis et une base de données stockée sous PGAdmin. Il m'a également servi lorsque que nous avons débuté le cas d'étude avec mes tuteurs et Mme Nathalie Saint-Geours.

f) BOUML

Pour terminer, j'ai évidemment utilisé BOUML qui était le cœur du stage, car de lui découlait l'intégralité des étapes à effectuer. En ayant déjà discuté assez longuement dans la partie analyse, je ne le représente pas ici.

Une fois l'ensemble des outils déterminés, j'ai dû choisir le langage de programmation dans lequel j'allais implémenter les parseurs. Aucune contrainte n'était imposée par mes tuteurs sur ce point-là.

2. Langages utilisés

a) Java

Java était totalement approprié à la problématique du stage. Celui-ci était en parfaite adéquation avec Eclipse. De plus, c'est avec ce langage que je possédais le plus d'affinités, et une des librairies mises à disposition convenait parfaitement pour l'implémentation des parseurs. En effet, la bibliothèque JDOM (Java Document Object Model) permet de

manipuler des fichiers basés sur le langage XML de manière rapide grâce aux nombreuses fonctions proposées. Or, XMI n'est autre qu'un format d'échange basé sur XML, donc compatible avec JDOM.

b) XML

Le langage XML (Extensible Markup Language) était quant à lui une contrainte imposée par la problématique du stage en elle-même.

En effet, le fait de devoir travailler sur des fichiers XMI rendait inévitable l'utilisation de XML. Cependant, ce dernier m'a été pratique pour réaliser un autre type de fichier appelé XSD. Un XSD permet de définir puis de valider la structure d'un fichier XML (dans notre cas, XMI), en fonction des règles de structure elles-mêmes définies dans ce XSD. Or, les XSD sont eux aussi écrit selon la syntaxe XML.

Une alternative était possible, en passant par une DTD (Document Type Definition), mais il n'était pas possible de définir une structure avec autant de rigueur que ne le propose XSD. Par exemple, avec une DTD, il est impossible de :

- Spécifier un ordre d'apparition des éléments fils
- Spécifier le nombre d'occurrences d'un élément
- Gérer très précisément les types de données contenus par les éléments et les attributs

Contrairement à la norme DTD, XSD respecte le format XML et a été approuvé par le W3C.

Nous avons désormais entre nos mains tout le nécessaire pour débuter l'étape suivante consistant à mettre en place une solution pour répondre à notre problématique.

IV. Solution apportée

1. Rendre BOUML opérationnel

Pour rappel, la version utilisée pour le stage est la 4.23, dernière version stable libre. Depuis, BOUML est commercialisé et en est actuellement à la version 6.6.

BOUML constitue le cœur du projet. Cependant, comme tout logiciel de modélisation UML libre, ce dernier souffrait de quelques défauts non négligeables à prendre en considération et à corriger. C'est pour cette raison que la première étape a consisté à paramétrer correctement le logiciel afin qu'il réponde parfaitement à nos besoins.

a) Personnalisation des fichiers de configuration

Un point noir qu'il fallait absolument résoudre concernait l'édition des attributs. En effet, comme l'ensemble des autres logiciels testés, il était impossible de spécifier quel attribut devait avoir le rôle d'identifiant ou quel attribut pouvait avoir la valeur « NULL ». Or, ces informations étaient essentielles pour passer du MCD au MRD. Dans le MRD, l'identifiant joue le rôle de clé primaire de la table en question, donc un élément primordial. Quant à la valeur « NULL », cela signifie qu'une fois la base de données implémentée, l'attribut en question aura la possibilité de ne se voir affecter aucune valeur, d'où le « NULL ». Malheureusement, nativement, BOUML ne proposait pas d'affecter aux attributs de telles informations. Cependant, la notion de stéréotype était parfaitement gérée par BOUML, nous permettant de contourner ce problème assez simplement.

i. Personnalisation des stéréotypes pour les attributs

Dans UML, les stéréotypes sont un mécanisme d'extensibilité permettant d'étendre le vocabulaire de l'UML, afin de créer de nouveaux éléments. Généralement, ils sont représentés ainsi : <<stéréotype>>. De plus, les stéréotypes peuvent être appliqués à des attributs, idéal pour notre cas. Initialement, voilà à quoi ressemble la fenêtre d'édition d'un attribut dans BOUML.

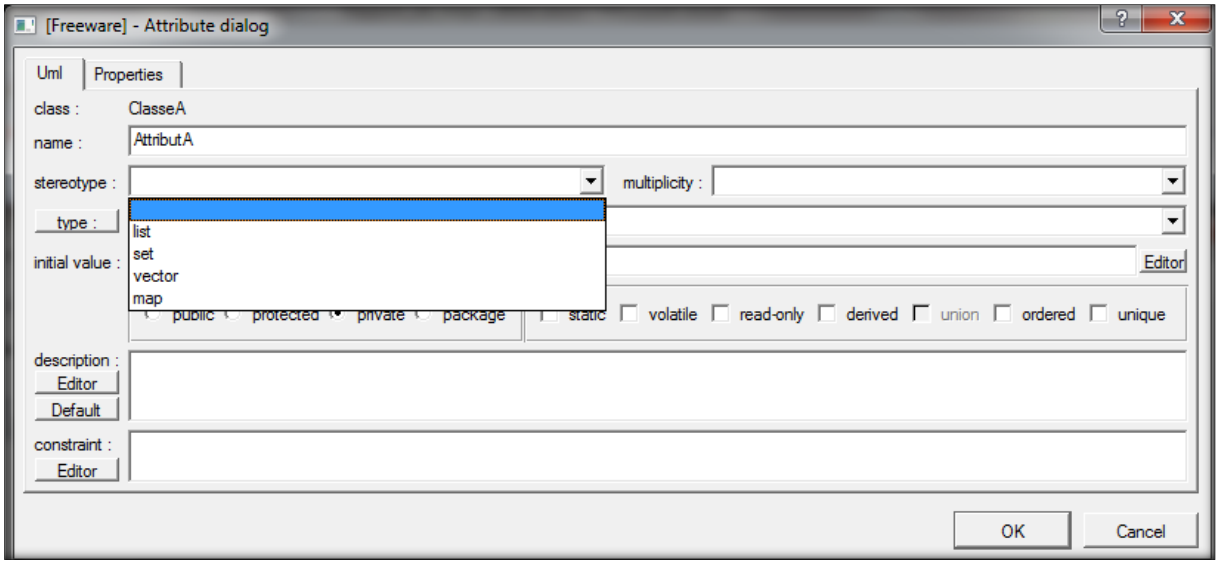

Figure 17 : Fenêtre d'édition initiale d'un attribut

Comme le montre la figure 17, initialement aucun champ ne permet de spécifier le fait qu'un attribut soit identifiant ou puisse être « NULL ». Cependant, il est possible de modifier le fichier de configuration proposant les stéréotypes proposés par défaut dans la liste déroulante. De base, les 4 stéréotypes suggérés par BOUML que sont <<list>>, <<set>>, <<vecor>> et <<map>> nous sont inutiles. Mais ceci n'est en aucun cas rédhibitoire car via les menus de BOUML nous pouvons accéder au fichier des stéréotypes et l'éditer comme nous le souhaitons. Il est nommé « stereotypes » et est généré à la création d'un nouveau projet. Après modification, voici la nouvelle liste proposée à l'utilisateur.

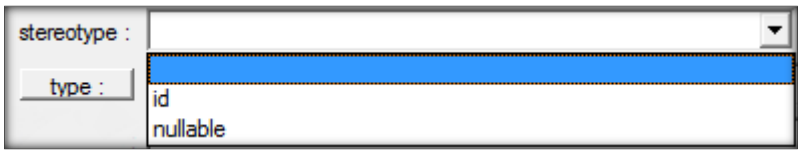

Figure 18 : Nouveaux stéréotypes

Nous avons donc deux stéréotypes nommés :

- **Id** : indique qu'un attribut sera identifiant donc à terme clé primaire dans le MRD.
- **Nullable** : l'attribut en question peut avoir la valeur « NULL » dans la base de données.

Un même attribut ne pourra pas se voir attribuer les deux stéréotypes en même temps, la première raison étant que la liste impose seulement zéro ou un choix, et la seconde concerne la sémantique même : si l'attribut est clé primaire alors il doit obligatoirement recevoir une valeur, donc ne peut pas être « NULL ».

Le second problème concernait toujours les attributs, mais cette fois sur les types proposés.

ii. Personnalisation des types pour les attributs

La figure ci-dessous indique les types proposés par BOUML pour les attributs.

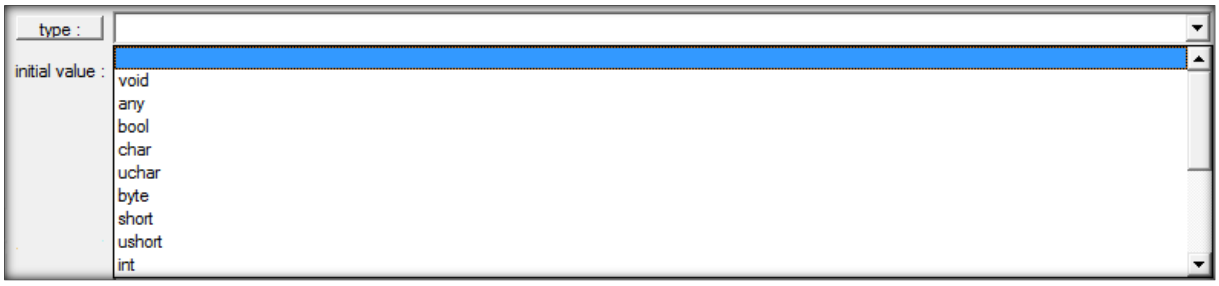

Figure 19 : Types proposés de base pour les attributs

La plupart des types proposés par défaut sont utiles, mais certain comme « any » et « byte » ne le sont pas dans notre contexte, et plus important encore, des types manquent à l'appel, tels que « date » et plus spécialement les types spatio-temporels cités précédemment. Pour répondre à ce besoin, la même procédure a été réalisée, en modifiant cette fois le fichier nommé « generation_settings ». Voici maintenant un aperçu de la nouvelle liste que l'utilisateur verra.

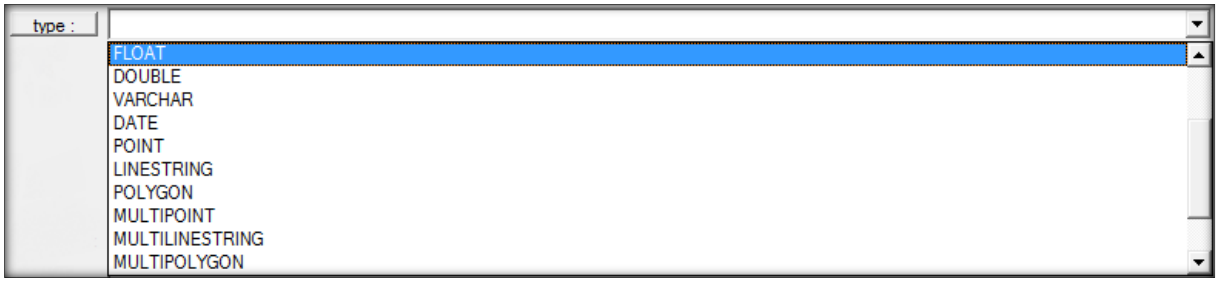

Figure 20 : Nouveaux types proposés pour les attributs

Comme nous le voyons ici, les types spatio-temporels sont proposés à travers « POINT », « LINE », »POLYGON » et les « MULTI » et le type « DATE » a été rajouté.

A la création d'un nouveau projet, il suffira donc d'importer les deux fichiers de configuration pour que BOUML soit complètement opérationnel.

En modifiant ainsi les fichiers de configuration, nous sommes sûrs que les utilisateurs ne se tromperont pas en choisissant le stéréotype et le type. En effet, nous pouvions laisser les fichiers tels quels et indiquer dans le manuel utilisateur d'écrire eux-mêmes le type ou stéréotype désiré car les listes nous laissent libres de rajouter ce que nous souhaitons simplement en l'écrivant. Evidemment cette méthode n'était en aucun cas convenable du simple fait que se tromper dans l'orthographe ou faire une faute de frappe arrive très fréquemment, et nous ne faisons pas forcément attention.

Nous avons également pu utiliser cette technique car toutes les informations dont nous avions besoin étaient conservées lors de l'export du MCD au format XMI. La

prochaine étape consiste justement à établir un fichier ayant pour rôle de valider le MCD créé par l'utilisateur, en se basant sur le format XMI généré.

b) Mise en place d'un XSD

Désormais, BOUML est correctement paramétré et prêt à l'emploi. Cependant, rien ne garantit que les Modèles Conceptuels de Données réalisés ne comportent pas d'erreurs dans leur structuration. Celles-ci peuvent provenir de l'utilisation d'un élément non géré car inutile pour le MCD. Les figures ci-dessous indiquent les éléments gérés ainsi que leur rôle.

Figure 21 : Barre de création du MCD dans BOUML

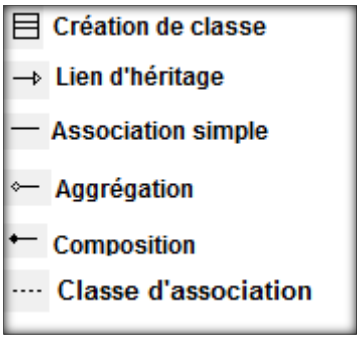

Figure 22 : Rôle des différents boutons gérés

Sachant que chaque élément ajouté possède au format XMI ses propres balises ainsi que ses propres attributs, insérer des éléments qui ne sont pas gérés est inutile et pourrait même causer des problèmes lors de l'analyse du fichier dans les parseurs. C'est la raison pour laquelle nous avons décidé de mettre en place un fichier XSD pour garantir un niveau de qualité sans bruit inutile. Le bruit est une notion simple à comprendre, correspondant à des éléments superflus et perturbant la compréhension du fichier.

Il est important de noter que la version retenue de XMI est la plus récente proposée par BOUML, étant la 2.1 couplée avec la version 2.3 d'UML. C'est donc en analysant de manière rigoureuse le format XMI de multiples MCD créés que j'ai pu établir la structure type d'un fichier XMI généré. La façon de procéder a été la même que celle expliquée précédemment, lorsque je comparais le format XMI de BOUML avec celui de Modelio.

J'ai examiné en détails la retranscription de tous les éléments gérés :

- Classes
- Classes d'association

- Attributs (y compris les stéréotypes et les types)
- Associations simples
- Agrégations
- Compositions
- Héritages
- Enumérations

De cette analyse j'ai pu établir toutes les règles de structuration dans un fichier XSD écrit au format XML. Plus précisément, j'ai réalisé deux XSD :

- Le premier vérifiant la structure du fichier exporté par BOUML (de MCD à MRD)
- Le second vérifiant la structure générée par le premier parseur (de MRD à SQL)

Voici un extrait du premier fichier XSD.

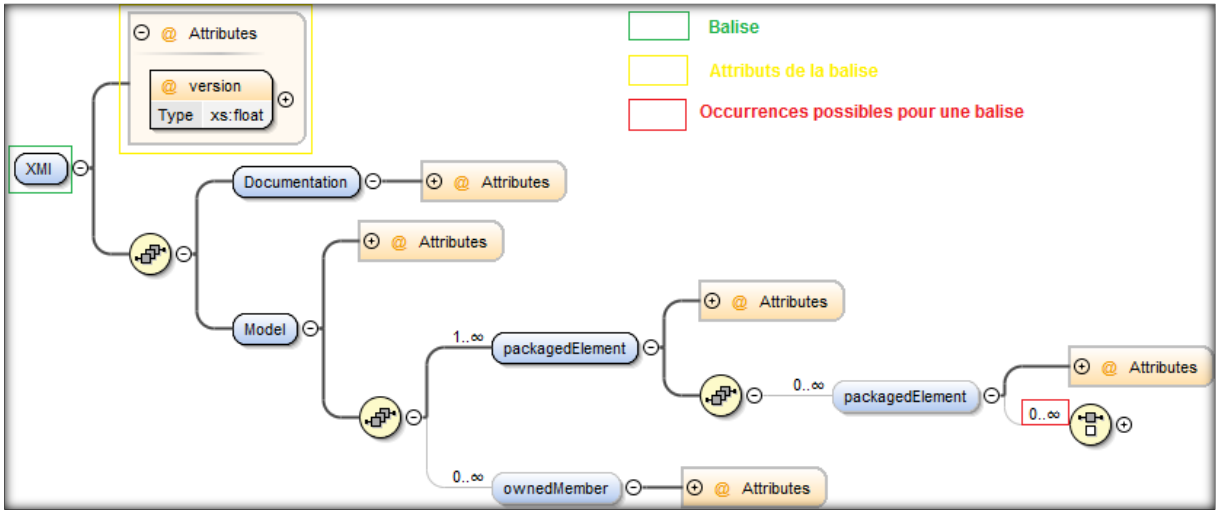

Figure 23 : Extrait de l'XSD pour passer du MCD au MRD

Cette figure représente le XSD ouvert sous le logiciel Oxygen. Nous comprenons ici la force de ce logiciel, car outre le fait de pouvoir éditer un XSD sous la forme XML, ce dernier nous propose de l'éditer sous cette forme graphique, en ajoutant par de simples boutons des balises et attributs. La racine se trouve à gauche, représentée par la balise « XMI », possédant un seul attribut nommé « version », avec pour valeur possible un float uniquement. Il est simple de comprendre par la suite que la balise « XMI » possède deux balises filles nommées « Documentation » et « Model », et ainsi de suite…

En réalisant un XSD, nous serons ainsi sûrs que la structure du MCD répondra bien aux attentes, et que le parseur pourra jouer son rôle efficacement. Tous les préparatifs nécessaires étant réalisés, nous pouvons débuter l'implémentation du premier parseur.

2. Premier parseur : MCD to MRD

Comme expliqué dans la première partie de ce rapport, le rôle du premier parseur est de transformer le Modèle Conceptuel de Données en Modèle Relationnel de Données par le biais de sa représentation sous le format XMI.

a) Les règles de transformation

Le passage du MCD au MRD se fait grâce à l'utilisation de transformations standards à appliquer en fonction des différents cas de figures rencontrés. Ce travail nécessite donc de connaître ces règles. C'est la raison pour laquelle une analyse en amont du parseur a été réalisée, consistant à établir la liste la plus exhaustive possible concernant les cas de figures que nous pouvons potentiellement rencontrer dans un MCD. Cette analyse devait prendre en compte une multitude de paramètres, tels que :

- ✓ La présence ou non d'une classe d'association
- ✓ Le type d'association (association simple, agrégation, composition, association réflexive, héritage)
- ✓ Les multiplicités présentes à chaque extrémité de l'association

La combinaison des différents paramètres listés engendre une transformation spécifique pour passer au Modèle Relationnel de Données. Voici un exemple basique présenté à travers la figure ci-dessous.

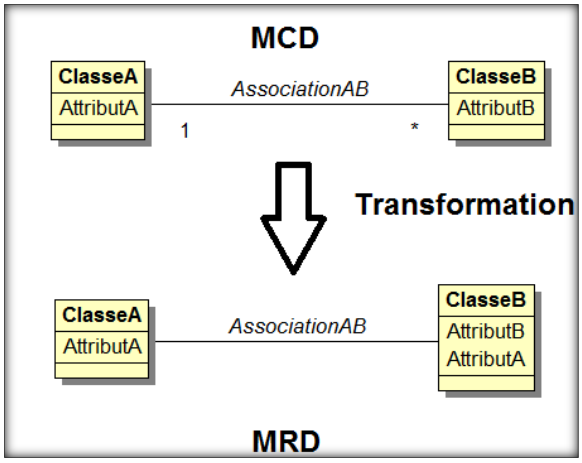

Figure 24 : Exemple de transformation

Ici, il n'y a pas de classe d'association, l'association « AssociationAB » est simple et la multiplicité est « 1-* ». De plus, « AttributA » et « AttributB » possèdent le stéréotype « id ». Dans ce cas, au passage dans le MRD, tous les attributs stéréotypés « id » présents dans « ClasseA » (uniquement AttributA ici) vont être dupliqués dans « ClasseB », devenant ainsi clé étrangères.

Le tableau suivant récapitule les règles fondamentales à appliquer, en se basant sur cette figure :

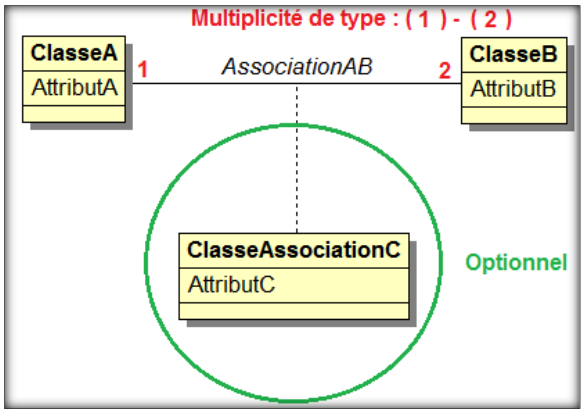

Figure 25 : Exemple de transformations basiques

Classe d'association / Multiplicités	Avec	Sans
(x,n) – (1,1) (Avec x≠n)	Les identifiants de « ClasseB » deviennent clés étrangères dans « ClasseA », les attributs de « ClasseC » sont migrés dans « ClasseA » et « ClasseC » est supprimée.	Les identifiants de « ClasseB » deviennent clés étrangères dans « ClasseA ».
(x,n) – (x,n) (Avec x≠n)	« ClasseC » devient une classe à part entière et récupère les identifiants de « ClasseA » et « ClasseB », devenant clés primaires et étrangères.	Création d'une nouvelle classe prenant les identifiants de « ClasseA » et « ClasseB » devenant clés primaires et étrangères.

Figure 26 : Règles de transformations fondamentales

Bien entendu, ce tableau ne présente qu'un extrait des transformations possibles. Ce dernier ne comprend pas les agrégations, compositions et spécialisations / généralisations. A travers ce tableau, le but est de montrer qu'un choix avait été pris concernant les multiplicités de type (0,1) – (0,1) et (x,n) – (x,1). En effet, deux écoles existent concernant ces cas de figures : l'une considère que dès qu'une des multiplicités a pour valeur maximale « 1 », alors la création d'une nouvelle classe est inutile, et qu'il suffit de faire migrer les identifiants d'une classe à l'autre. Pour cette école, la présence de (0,1) – (0,1) laisse donc le choix au concepteur de faire migrer les identifiants dans les deux sens.

L'autre école considère que ces types de multiplicités nécessitent la création d'une nouvelle classe, et seule la présence d'une multiplicité de type (1,1) – (x,n) permet de faire migrer les identifiants. Au vu du tableau, nous comprenons bien que la solution que nous avons retenue est la seconde. La seule différence est la gestion des valeurs « null » lors de l'insertion de données dans la base de données.

Notre volonté était d'éviter au maximum la gestion de données pouvant être potentiellement « null », car cela peut vite générer de l'incompréhension et est généralement considéré comme une solution peu propre.

Cependant, une multiplicité n'est pas présentée dans le tableau, cette dernière étant (1) – (1).

i. Cas spécial : multiplicité de type (1) – (1)

Le premier cas de figure pour lequel nous laissons l'utilisateur choisir est celui présenté ci-dessous.

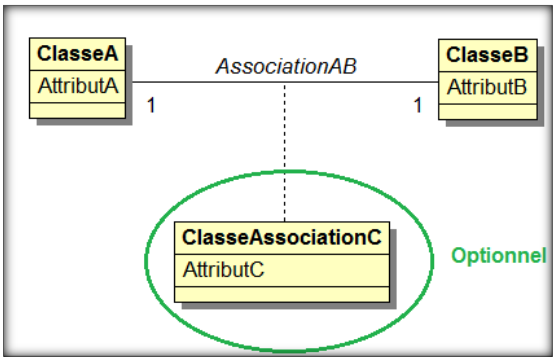

Figure 27 : Cas de multiplicité 1-1

Pour une question de sémantique, il est proposé à l'utilisateur de choisir entre :

- Fusionner les deux classes en une seule (y compris la classe d'association si elle est présente) : Simplifie la gestion des données en ne possédant plus qu'une seule classe.
- Garder les deux classes, et choisir quel identifiant sera migré dans l'autre classe (si la classe d'association est présente, ses attributs iront également dans la classe choisie) : Permet de conserver la sémantique introduite par la création des deux classes, en laissant séparer de manière visible les attributs.

L'autre cas de figure pour lequel le choix est laissé à l'utilisateur concerne l'héritage.

ii. Cas spécial : les liens de spécialisation / généralisation

Les règles standards de transformation prévoient 4 possibilités pour gérer la spécialisation / généralisation :

- ✓ Supprimer toutes les classes filles en faisant remonter au préalable leurs attributs dans la classe mère la plus haute
- ✓ Conserver toutes les classes en faisant migrer l'identifiant de la classe mère dans l'ensemble de ses classes filles (de profondeur n)
- ✓ Conserver toutes les classes en dupliquant l'ensemble des attributs de la classe mère dans toutes ses classes filles
- ✓ Ne conserver que les classes filles en ayant au préalable dupliqué l'ensemble des attributs de la classe mère dans toutes ses classes filles

Nous avons décidé de gérer uniquement les deux premières solutions listées. Les deux dernières solutions ne nous semblaient pas pertinentes car cela engendrait énormément de redondance.

La figure ci-dessous illustre les solutions retenues.

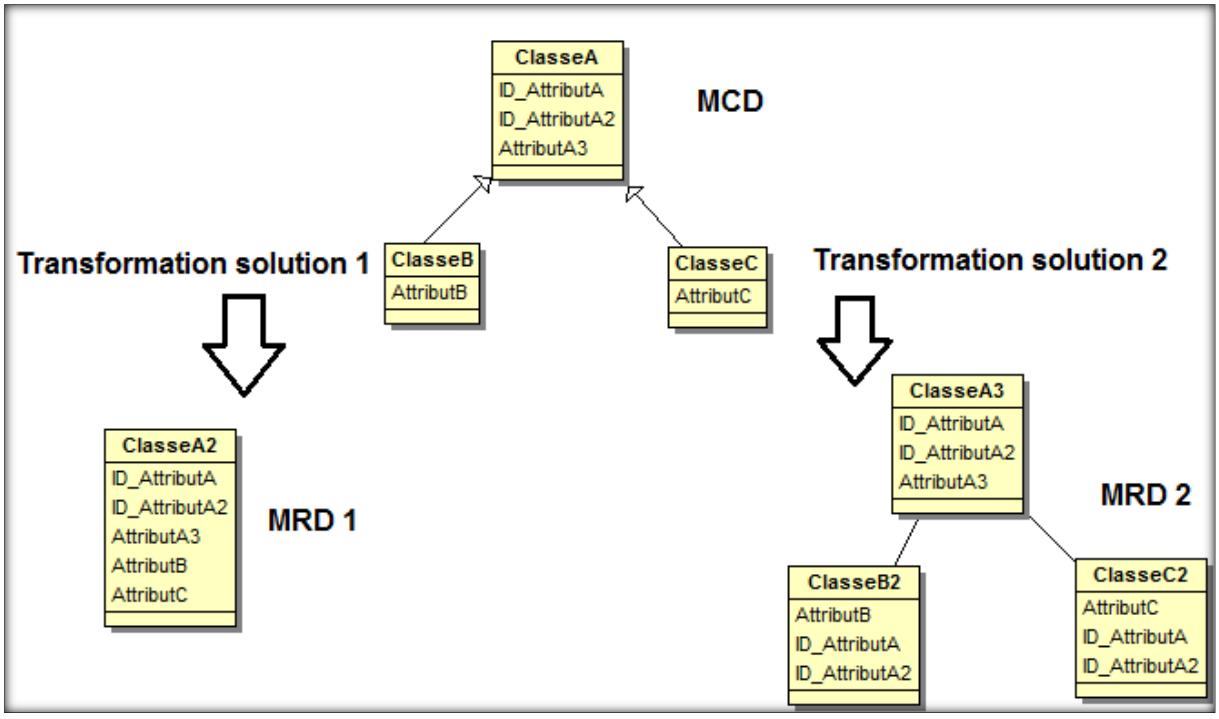

Figure 28 : Transformation héritage

Ce choix est laissé à l'utilisateur pour qu'il puisse personnaliser au maximum son Modèle Relationnel de Données, en le rendant le plus flexible possible. La première solution permet de simplifier la gestion des données en ne possédant plus qu'une seule classe, mais en perdant fatalement de la sémantique et obligeant à gérer la notion de valeur « NULL ».

La seconde solution est celle la plus souvent utilisée, car elle conserve toute la sémantique apportée dans le MCD et permet de naviguer simplement à travers les différentes classes.

Tous les types de transformations n'ont pas été évoqués ici, notamment la gestion des compositions, agrégations et associations réflexives, mais il est difficile de toutes les présenter de manière exhaustive car cela prendrait beaucoup de temps et n'apporterait pas grand-chose pour la suite. Mais pour la mise en place du parseur, j'ai bien évidemment pris en considération l'intégralité des cas de figures. C'est d'ailleurs durant toute cette phase d'analyse que j'ai utilisé PowerAMC pour observer son comportement face à des transformations peu évidentes. Il m'a également été utile pour comparer mes MRD obtenus avec ceux que lui-même générait en partant du même MCD. De cette façon, en me basant sur un outil ayant déjà fait ses preuves et étant réputé pour être stable, je pouvais vérifier simplement et rapidement d'où venaient mes erreurs s'il y en avait.

Maintenant que l'analyse des règles de transformations a été effectuée, nous pouvons passer à l'implémentation du parseur en lui-même.

b) Implémentation

Comme nous l'avons dit précédemment, le parseur a été réalisé en JAVA. Deux bibliothèques ont été indispensables pour sa réalisation :

- **JDOM** : bibliothèque OpenSource permettant de manipuler des fichiers XML (des fichiers XMI dans notre cas).
- **Swing** : bibliothèque intégrant les éléments graphiques nécessaires pour réaliser l'interface du parseur.

Cinq classes composent le parseur :

- **Parser_MCD_MRD** : classe jouant le rôle d'interface graphique du parseur.
- **TransformationMCDinMRD** : classe qui prépare le fichier XMI sur lequel nous allons travailler.
- **TransformationEntities** : classe appliquant les transformations d'entités à entités.
- **TransformationsAssociationsClasses** : classe appliquant les transformations dans le cas où une classe d'association est présente.
- **TransformationLegacy** : classe appliquant les transformations pour les liens d'héritage.

En dehors de ces cinq classes, un autre fichier est indispensable au bon fonctionnement du parseur, s'agissant du XSD nommé « Schema_XMI_MCD.xsd ». Voyons le déroulement de l'algorithme permettant de transformer le MCD en MRD.

i. L'algorithme : préparation du fichier XMI

Une fois le MCD terminé et exporté au format XMI, le premier parseur entre en jeu. Ce dernier est fourni via un JAR exécutable. A son lancement, l'interface graphique va

guider pas à pas l'utilisateur en lui précisant ce qu'il doit faire. Dans un premier temps l'utilisateur devra récupérer son fichier XMI via un explorateur de fichiers. A ce moment-là, le programme va créer une structure de dossiers comme indiqué ci-dessous.

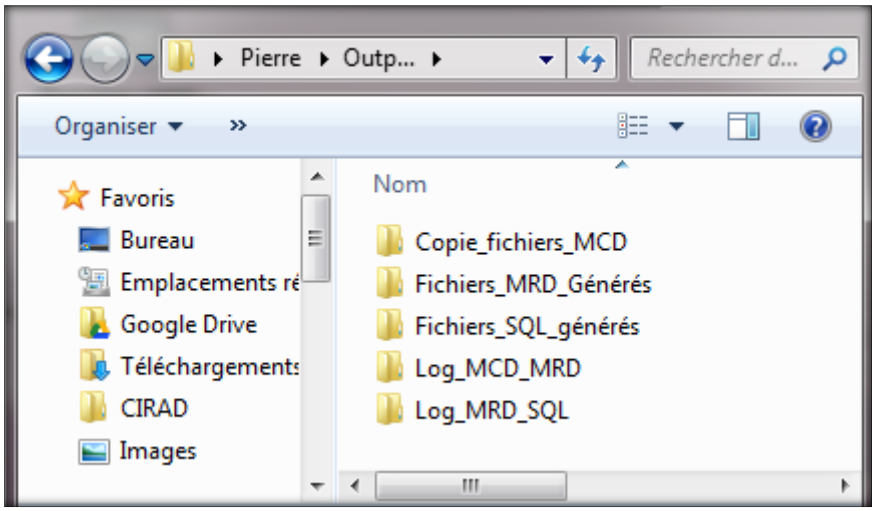

Figure 29 : Structure de dossiers générée

Comme l'indique le nom du premier dossier, une copie du fichier XMI choisi par l'utilisateur va être réalisée. C'est sur cette copie que l'analyse va être faite. En effet, le fichier de base ne sera ni utilisé par la suite ni modifié. Par contre, la copie va subir des transformations dans sa structure pour répondre aux règles fixées dans le XSD et surtout pour supprimer des éléments inutiles pour le parseur. Ceci est le rôle de la classe « TransformationMCDinMRD ». Cette dernière va donc copier le fichier XMI puis va effectuer les modifications suivantes :

- ✓ Suppression de certaines balises inutiles.
- ✓ Modification de l'espace de nom de chaque balise.

L'étape consistant à modifier l'espace de nom était très importante. Voici l'entête du fichier de base généré par BOUML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.3" xmlns:xmi
    <xmi:Documentation exporter="Bouml" exporterVersion="1.15"/>
    <uml:Model xmi:type="uml:Model" xmi:id="themodel" name="Exemple_Rapport">
```

Espace de nom xmi:

Espace de nom uml:

Figure 30 : Entête généré par BOUML

Comme l'indique les deux cadres, deux espaces de nom différents sont imbriqués, nommés respectivement « xmi » et « uml ». Seulement, lorsque j'ai mis en place le XSD, je n'ai pas réussi à définir comme règle que l'espace de nom « xmi » pouvait contenir un autre espace de nom, en l'occurrence « uml ». La raison est simple, car un fichier XSD ne peut pas gérer deux espaces de nom différents. Après recherche, il existe une solution consistant à importer un fichier XSD gérant l'espace « uml » dans un autre XSD gérant

l'espace « xmi ». Cependant, n'ayant pas trouvé le moyen d'imbriquer différents XSD gérant plusieurs espaces de nom, j'ai opté pour la solution citée plus haut : modifier l'espace de nom de manière récursive pour chaque balise pour ne plus avoir que du « xmi ».

Mentionner cette étape peut sembler être sans importance, mais celle-ci était tout simplement cruciale pour la suite et modifier les espaces de nom a été plus difficile qu'il n'y parait. En effet, JDOM offrait une fonction permettant de le faire, mais dès qu'elle était utilisée, la fonction provoquait des effets indésirables sur la structure du fichier, invalidant le XSD.

A noter que le dossier nommé « Log_MCD_MRD » contiendra la trace d'exécution du premier parseur pour chaque lancement. Dans ce dernier sera créé systématiquement un fichier à la date et l'heure du jour dans lequel est redirigé tous les « System.out.println( ) » ainsi que les messages d'erreur complets. Ceci n'est pas forcément utile pour une personne non-initiée mais facilite grandement le débogage et fournit une vision du déroulement de l'algorithme dans son ensemble.

A ce stade, la copie est préparée et prête à être analysée, et en parallèle le fichier XMI qui correspondra au MRD a été créé.

ii. L'algorithme : application des règles de transformation

Après avoir récupéré son fichier XMI, l'utilisateur choisira la solution qui lui convient le mieux pour les deux cas de figures déjà cités précédemment (pour la multiplicité (1)–(1) et les liens de spécialisations / généralisations). Maintenant, l'utilisateur n'a plus qu'à attendre (sauf cas spéciaux nécessitant son interaction avec l'interface) que le MRD soit généré dans le dossier « Fichiers_MRD_Générés ». C'est ici que les trois classes nommées « TransformationEntities », « TransformationLegacy » et « TransformationAssociationClasses » entrent en jeu, mais pas dans n'importe quel ordre. En effet, il a fallu hiérarchiser les différentes transformations à effectuer, car si les transformations sont réalisées dans un ordre totalement aléatoire, le MRD obtenu à la fin sera très probablement erroné. Voici donc les associations à gérer, de la plus prioritaire à la moins prioritaire.

1. Héritages / compositions
2. Associations simples / agrégations / associations réflexives
3. Associations comprenant une classe d'association

L'héritage et la composition sont les deux associations les plus prioritaires car elles ont une particularité que les autres n'ont pas. Lorsqu'un identifiant migre d'une classe à l'autre, celui-ci devient clé étrangère comme pour le reste des autres transformations, mais cet identifiant va également être clé primaire dans l'autre classe. Ceci va donc avoir des répercussions sur le reste des associations, car cet identifiant pourra migrer dans d'autres classes plus lointaines. Pour mieux comprendre, voici un exemple concret.

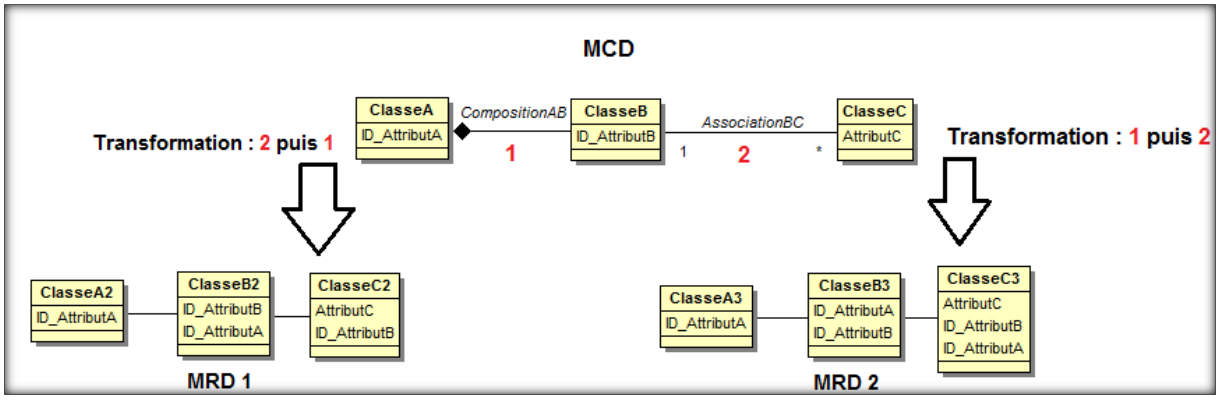

Figure 31 : Importance de l'ordre d'exécution

Cet exemple illustre bien le fait que l'ordre d'exécution est primordial, car sinon nous pouvons nous retrouver avec des MRD peu adaptés. Ici, c'est « MRD 2 » qui est correct car la composition est réalisée avant l'association simple et donc « ID_AttributA » est bien reporté dans « ClasseC ».

C'est donc « TransformationLegacy » qui va être exécutée en première. A partir de là, une nouvelle notion capitale fait son apparition, s'agissant des requêtes XPATH. JDOM propose nativement d'utiliser le système de requêtes XPATH, système sur lequel repose l'intégralité du parseur. Une requête XPATH permet de récupérer des éléments précis d'un fichier XML en l'interrogeant en posant des conditions. Par exemple, ici le but est de capter toutes les classes possédant un lien d'héritage dans le MCD ainsi que toutes les classes possédant un lien de composition. Ceci se traduit par la figure suivante.

```
XPath getComposition = XPath.newInstance("//xmi:ownedAttribute[@aggregation='composite']");
List results2 = getComposition.selectNodes(TransformationMCDinMRD.rootMCD);
                    REQUETE XPATH RECUPERANT LES LIENS DE COMPOSITION


XPath getLegacy = XPath.newInstance("//xmi:packagedElement/xmi:generalization");
List results = getLegacy.selectNodes(TransformationMCDinMRD.rootMLD);
                    REQUETE XPATH RECUPERANT LES LIENS D'HERITAGE
```

Figure 32 : Exemple de requêtes XPATH

Comme le montre cette figure, il est très simple d'interroger le fichier XML via XPATH. La première requête se traduit par : « récupère l'ensemble des balises nommées ownedAttribute possédant l'attribut aggregation ayant pour valeur composite ».

La seconde se traduit par : « récupère l'ensemble des balises nommées generalization qui sont imbriquées dans une balise packagedElement ».

Une fois ces associations en notre possession, il nous suffit de naviguer à travers le fichier et de réaliser les transformations nécessaires. Dès qu'une transformation est réalisée, son résultat est directement retranscrit dans le fichier représentant le MRD, qui est donc construit pas à pas.

Cependant, une particularité supplémentaire est venue perturber le déroulement de « TransformationLegacy ». Au sein même de la composition et de l'héritage existe une hiérarchie complexe. Prenons comme exemple la figure ci-dessous.

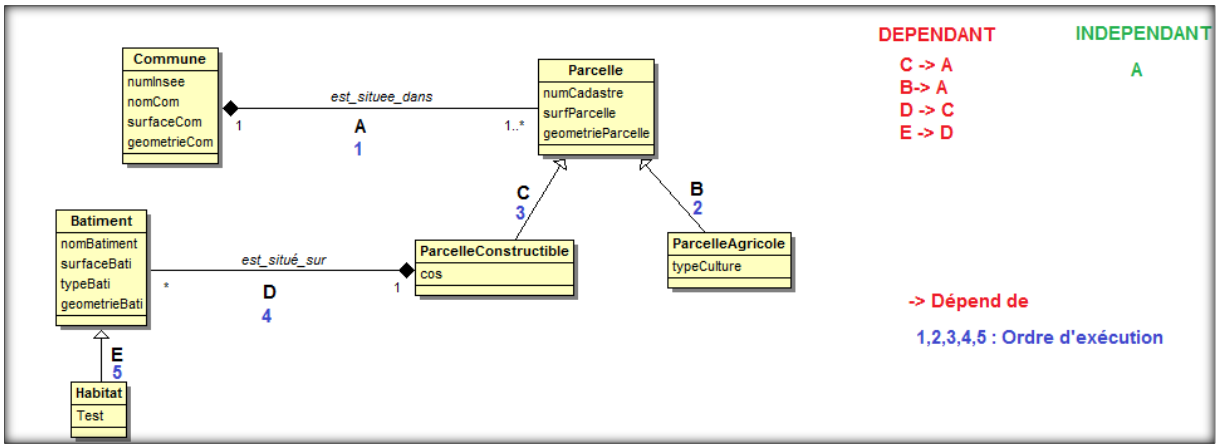

Figure 33 : Hiérarchie composition et héritage

Ceci est un extrait d'un MCD fourni par mon tuteur Michel Passouant. Nous voyons que 4 associations sur 5 dépendent d'autres associations. Dans « TransformationLegacy », j'ai donc utilisé deux structures de données :

- Une liste déclarée « `List<String> independant ;` »
- Une HashMap déclarée « `HashMap<String, List<String>> dependant ;` »

« independant » contiendra donc au départ les identifiants de toutes les associations qui ne dépendent d'aucunes autres pour pouvoir être exécutée.

« dependant » contiendra les couples clé/valeur suivant :

- Clé : identifiant de l'association qui est dépendante.
- Valeur : liste des identifiants des associations dont dépend la clé.

Donc, au fur et à mesure que les associations indépendantes seront exécutées, celles-ci seront supprimées de la liste « independant », et seront également supprimées de la liste de dépendances.

Par exemple, dans le cas ci-dessus, seule l'association A peut être exécutée. Une fois fait, A disparaîtra de « independant », et sera également supprimée dans la liste de dépendances de B et C. De cette manière, B et C deviendront indépendants car leur liste de dépendances sera vide, et pourront être exécutées à leur tour jusqu'à ce que toutes les associations soient exécutées. Voici le déroulement de la classe « TransformationLegacy ». Afin de clarifier mes propos, voici ce que deviendront les variables « dependant » et « independant ». L'extrait de code correspondant à cet algorithme est fourni dans l'annexe technique.

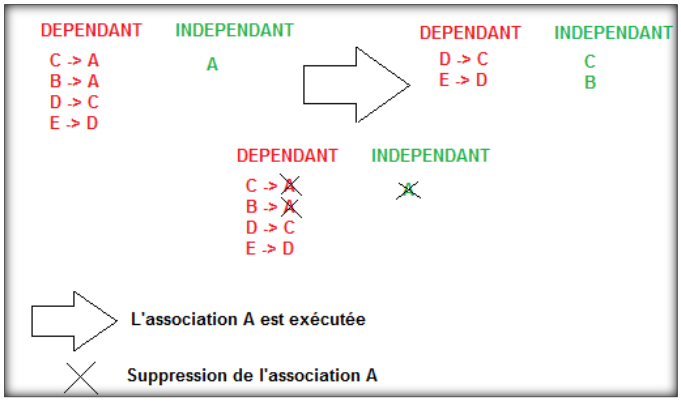

Figure 34 : Déroulement de l'algortihme

Vient ensuite l'exécution de la classe « TransformationEntities » qui est beaucoup plus simple étant donné que les liens de composition et d'héritage ont déjà été traités. De la même façon, une requête XPATH va récupérer l'ensemble des associations binaires pour lesquelles il n'y a pas de classes d'associations. Pour chacune de ces associations, il nous sera possible de récupérer les multiplicités présentes aux extrémités gauche et droite de l'association, et d'appliquer les transformations qui s'imposent (migration d'identifiant, création d'une nouvelle classe, fusion…). Bien entendu, une transformation ne concernant que les attributs ayant pour stéréotype <<id>>, il faut les récupérer en parcourant le fichier XMI à l'aide de requêtes XPATH, ou en parcourant les attributs d'une classe et de vérifier pour chacun d'eux s'ils possèdent bien ce stéréotype.

Une multitude de manipulations sont nécessaires, mais il ne serait pas pertinent de toutes les détailler bien qu'elles soient toutes indispensables pour arriver au résultat final. Mais la logique reste toujours plus ou moins la même : naviguer dans le fichier XMI, récupérer les informations dont nous avons besoin et en déduire les transformations adéquates. Ce parseur n'aurait pas pu être réalisé sans tout le travail réalisé en amont consistant à analyser et comprendre le système de structuration proposé par BOUML.

Enfin vient le tour de la classe « TransformationAssociationsClasses » qui va s'occuper d'effectuer les transformations sur les associations possédant des classes d'associations. Celles-ci sont similaires aux précédentes à quelques détails près. Par exemple, lors d'une multiplicité (0,1)-(0,1), il est ici inutile de créer une nouvelle classe, il faut cependant indiquer que la classe d'association devient une classe à part entière, toujours en jouant sur le vocabulaire de XMI et en modifiant certaines balises, attributs et valeurs, sans oublier de faire migrer les identifiants nécessaires.

En parallèle du déroulement de ces algorithmes, une erreur peut être levée à n'importe quel moment, concernant :

- Un problème de structuration du MCD
- Un attribut ne possédant pas de type
- Une association n'ayant pas toutes ses multiplicités renseignées

Un message personnalisé pour chacune de ses erreurs apparaîtra pour aider l'utilisateur au maximum à trouver d'où vient le dysfonctionnement.

Cependant, si tout se déroule correctement, le fichier représentant le MRD sera stocké dans le dossier « Fichiers_MRD_générés ». Il est même possible de l'importer dans BOUML pour voir à quoi ressemble le MRD. Voici une illustration de l'interface graphique pour ce premier parseur.

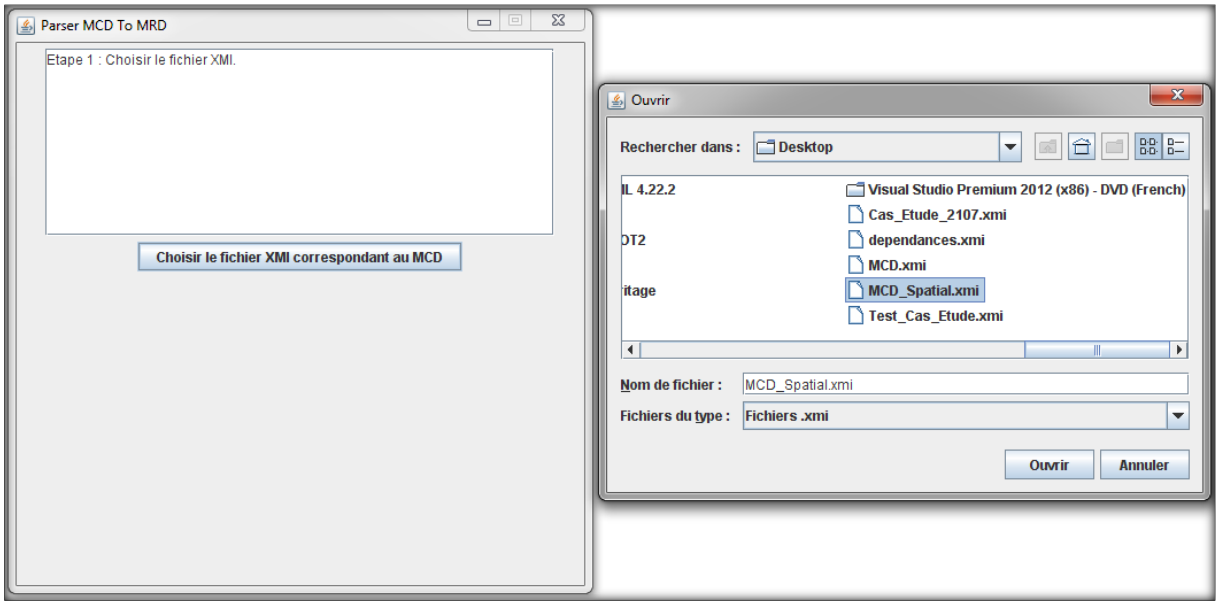

Figure 35 : Interface graphique du premier parseur

Pour plus d'illustrations et voir comment est proposé à l'utilisateur de gérer les cas spéciaux que sont les liens d'héritage et la multiplicité (1)-(1), je vous invite à regarder le manuel utilisateur fourni dans l'annexe technique.

Le fichier XMI obtenu représentant le MRD respecte la structure de base de BOUML, raison pour laquelle il nous est possible de l'importer de nouveau. Cependant, des éléments ont été supprimés, car rendus obsolètes pour le MRD tels que :

- Les informations concernant les multiplicités pour une association
- Les classes d'associations (soit supprimées soit devenues une classes)
- Les balises indiquant des liens d'héritage et de composition

Il ne reste donc au final que le strict minimum qui se résume aux classes avec leurs attributs. Cependant, j'ai dû enrichir le vocabulaire XMI de BOUML pour conserver des informations essentielles pour la suite. En reprenant la figure 33, l'attribut « numInsee » présent dans la classe « Commune » possède le stéréotype <<id>>. La classe « Parcelle » possédant un lien de composition vers la classe « Commune », la transformation qui s'impose est de dupliquer l'attribut « numInsee » dans « Commune. La figure ci-dessous présente dans le MRD comment est décrit « numInsee » dans « Commune » et dans « Parcelle ».

Figure 36 : Représentation d'un attribut dans deux classes différentes

Pour des soucis de clarté, j'ai supprimé des attributs de la balise « ownedAttribute » afin de ne conserver que ceux qui nous intéressent ici. Dans la classe « Commune », « numInsee » ne comporte que des informations basiques telles que son nom et son identifiant. En revanche, dans « Parcelle », j'ai dû ajouter des attributs (cadre marron) afin de conserver des informations très importantes pour la suite, sans lesquelles le passage au script SQL est impossible. Voici le rôle de ses balises :

- **foreignKeyAttribute** : nom de l'attribut référencé dans la classe « Commune ».
- **xmi :idref** : identifiant de l'attribut référencé dans la classe « Commune ».
- **foreignKeyEntity** : nom de la classe dans laquelle se trouve l'attribut référencé.

Ayant désormais en possession notre fichier XMI représentant le MRD, nous pouvons passer à la dernière étape consistant à utiliser le second parseur.

3. Second parseur : MRD to SQL

Nous approchons du but initial qui est de permettre à l'aide d'un Modèle Conceptuel de Données de fournir à l'utilisateur un script SQL directement importable dans des systèmes de gestion de bases de données tels que MS ACCESS, MySQL et PostGreSQL. La partie la plus difficile était de transformer le MCD en MRD, au vu des nombreuses transformations à apporter et des multiples algorithmes à mettre en place. En effet, le plus gros du travail a été réalisé dans le MRD car ce dernier a été construit de telle manière à simplifier au maximum la génération du script SQL, notamment grâce aux attributs ajoutés et expliqués quelques paragraphes plus tôt. Cependant, la difficulté dans cette dernière étape est qu'il fallait être capable de produire trois scripts SQL différents, en fonction des besoins de l'utilisateur. Chaque SGBD possède ses particularités syntaxiques, certains mécanismes sont gérés plus ou moins simplement voire pas du tout. Une seconde phase de recherche était donc nécessaire afin de prendre en main toutes les subtilités proposées par ces différents SGBDs.

a) Interface et structure du programme

Contrairement au premier parseur, l'utilisateur n'aura ici pas de choix à faire, si ce n'est le format SQL désiré. Il devra évidemment au préalable récupérer son fichier MRD généré précédemment via un explorateur de fichiers et nommer le script SQL qui sera fourni en sortie. Voici l'interface du second parseur.

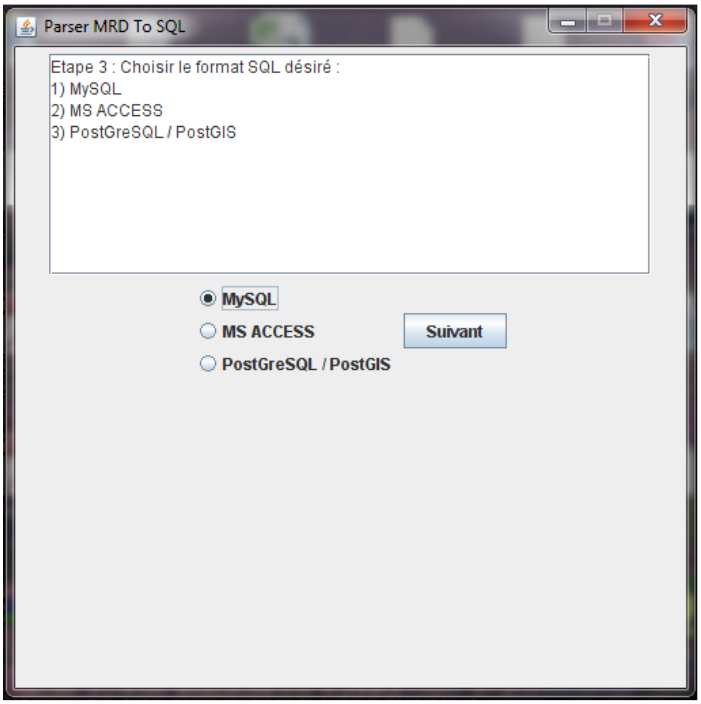

Figure 37 : Interface du second parseur

Le parseur « MRDtoSQL » est également composé de cinq classes :

- **Parser_MRD_SQL** : classe jouant le rôle d'interface graphique du parseur
- **SQL_Abstract** : classe abstraite regroupant toutes les fonctions communes pour générer des scripts MySQL, PostGreSQL et MS ACCESS.
- **MySQL_Generator** : classe générant le script MySQL (Hérite de SQL_Abstract)
- **MS_ACCESS_Generator** : classe générant le script MS ACCESS (Hérite de SQL_Abstract)
- **PostGreSQL_PostGIS_Generator** : classe générant le script PostGreSQL (Hérite de SQL_Abstract)

Il est délivré sous forme de JAR exécutable, comme le parseur « MCDtoMRD » et contient également un fichier supplémentaire correspondant au XSD. Ce dernier sert uniquement de sécurité, mais si le passage du MCD au MRD s'est déroulé correctement, il n'y a normalement pas de raison pour qu'une faille dans la structure soit détectée à ce stade-là. Ce dernier est cependant apprécié dans le cas où l'utilisateur souhaiterait utiliser ce second parseur sur un fichier XMI représentant un MCD, auquel cas un message d'erreur apparaîtrait indiquant qu'il s'est trompé.

Au lancement, le même mécanisme est utilisé : une copie du fichier représentant le MRD est réalisée, copie sur laquelle nous allons travailler. En reprenant la figure 28, le script sera généré dans le dossier « Fichiers_SQL_Générés » et un fichier de la trace d'exécution sera produit dans le dossier « Log_MRD_SQL ». Avant de débuter l'explication des différents algorithmes, il est important de mentionner quelque chose dont je n'ai pas encore parlé jusqu'ici.

i. Les métadonnées

Cette notion est en effet primordiale pour le projet. Le but est évidemment de fournir à l'utilisateur un script SQL prêt à l'emploi, mais pas seulement. Pour rappel, le stage s'inscrit dans le contexte du « Patrimoine Numérique ». Ce dernier prévoit la réutilisation des données issues de travaux de recherche antérieurs. Pour ce faire, documenter la base de données est indispensable pour garder une trace permettant à une personne lambda de comprendre en quelques minutes la logique instaurée et le rôle de chaque donnée. Ceci est parfaitement géré par BOUML. Lors de l'édition d'une classe ou d'un attribut, un champ nommé description est disponible. Ce qui est écrit dedans sera conservé dans le XMI généré de BOUML, et conservé par mon premier parseur dans le XMI représentant le MRD.

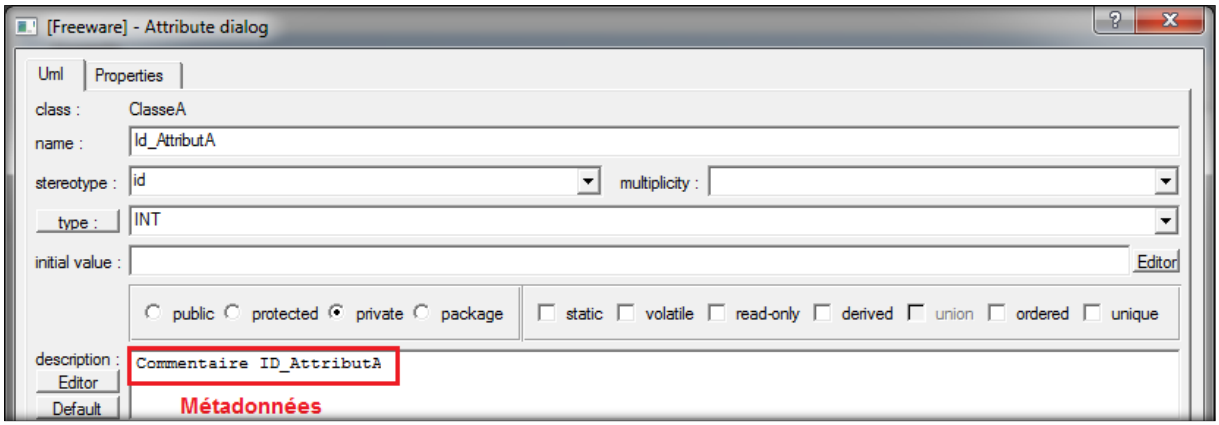

Figure 38 : Métadonnées inscrites dans BOUML

C'est seulement lors de l'exécution du second parseur que cette information sera traitée et utilisée.

b) Script MySQL

J'ai débuté la génération de script par celui au format MySQL, car c'est tout simplement celui avec lequel j'étais le plus familier. J'ai utilisé en parallèle PhpMyAdmin pour tester la syntaxe que je générais, car ce dernier permet d'importer très simplement des scripts. L'algorithme en soi est presque identique pour les trois formats. Seules quelques différences sont notables, comme par exemple la gestion des métadonnées, les

contraintes de domaine ou les données spatio-temporelles. MySQL était de loin le plus simple à produire. Ici, l'algorithme consiste à :

1. Créer un fichier à extension « .sql » avec le nom choisi par l'utilisateur
2. Récupérer toutes les tables via une requête XPATH dans le fichier XMI
3. Créer toutes les tables au format MySQL en déclarant les clés primaires (attributs possédant le stéréotype <<id>>) en écrivant dans le fichier créé précédemment
4. Modifier les tables en y ajoutant les clés étrangères

Ces étapes sont exécutées dans la classe « MySQL_generator ». Tout se fait en parcourant le fichier XMI via des requêtes XPATH et en naviguant à travers celui-ci grâce aux identifiants de telle ou telle balise.

Voici ci-dessous un MCD représenté sous BOUML avec un extrait du script MySQL correspondant.

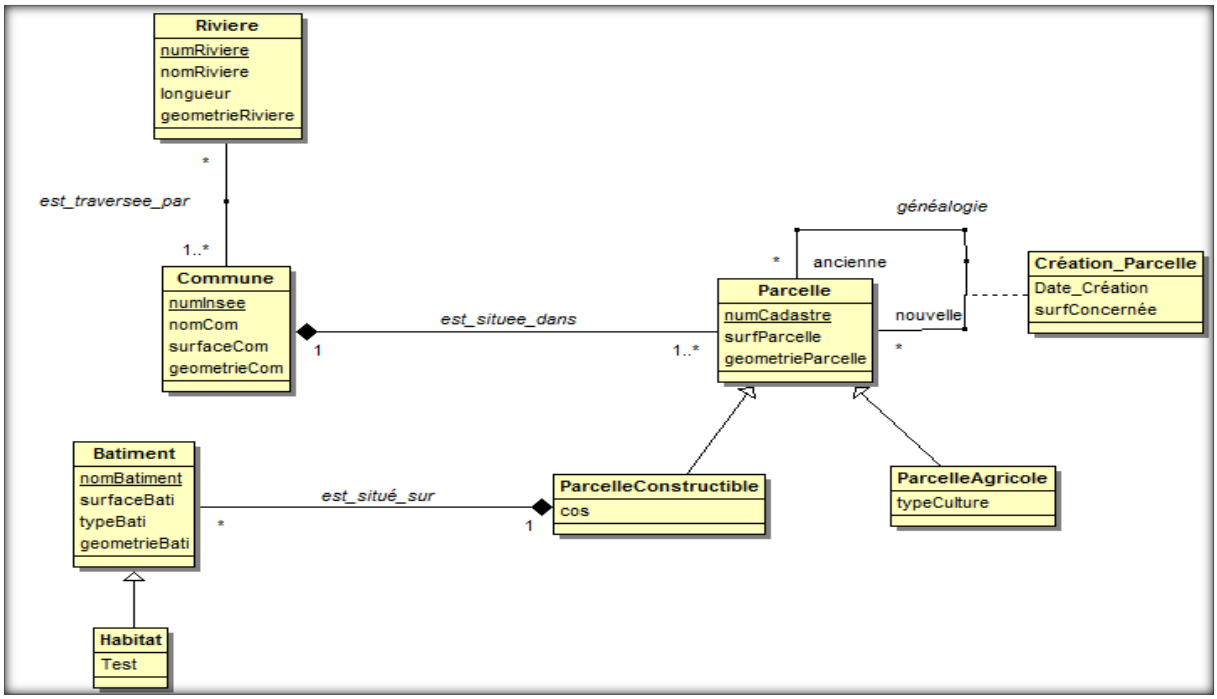

Figure 39 : Exemple de MCD

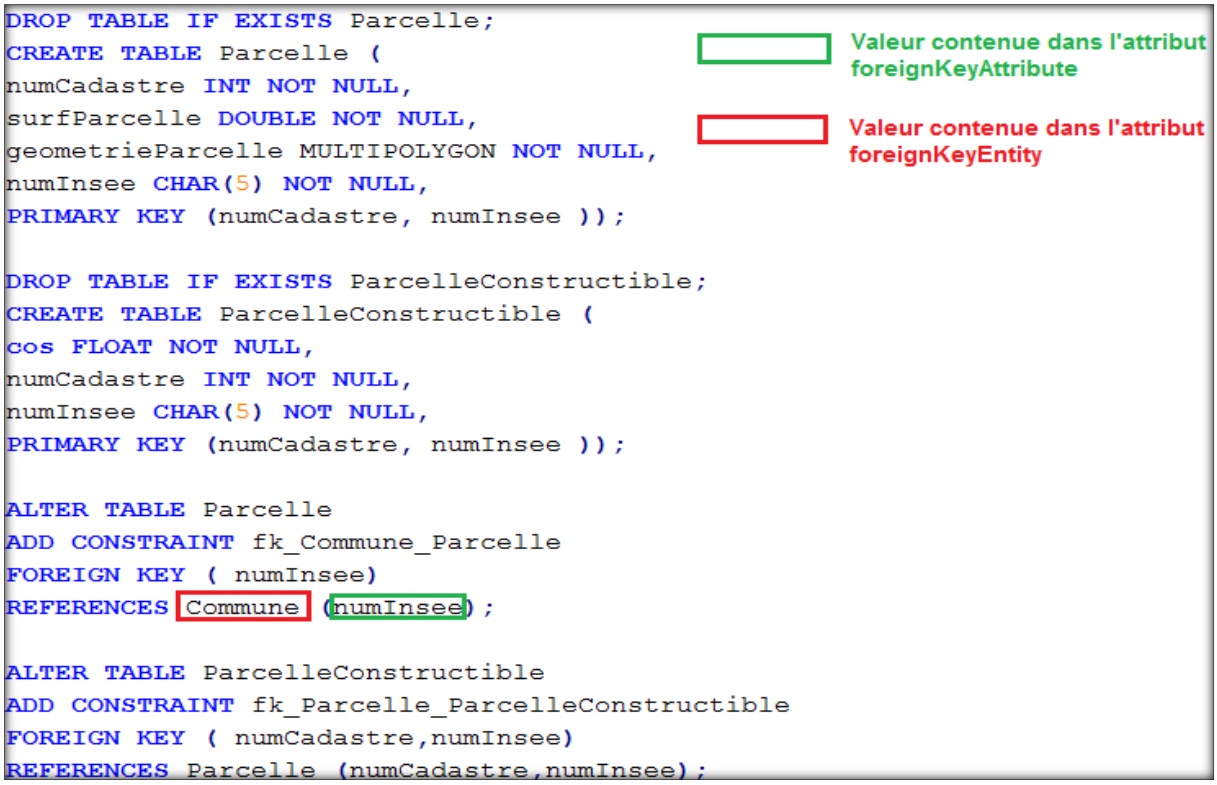

Figure 40 : Extrait script MySQL

Pour revenir aux attributs créés dans le MRD nommés « foreignKeyEntity » et « foreignKeyAttribute », ils prennent toute leur importance lors de la dernière étape consistant à ajouter les contraintes de clés étrangères, comme l'indique les cadres rouge et vert. Sans eux, rien ne précise que l'attribut « numInsee » présent dans la table « Parcelle » fait référence à l'attribut « numInsee » présent dans la table « Commune ».

Comme dit précédemment, les contraintes de domaine ne sont pas un problème pour le format MySQL. Une contrainte de domaine signifie qu'un attribut, en plus de son type attribué, possède des restrictions concernant les données qu'il peut contenir. Par exemple l'attribut « numCadastre » est assigné au type « INT », signifiant que « numCadastre » peut contenir des données de type entier. Si ce dernier avait été assigné au type « INT UNSIGNED », alors « numCadastre » n'aurait pu contenir que des données de type entier positif. Heureusement, MySQL prévoit le type « INT UNSIGNED » nativement, ce qui n'est pas le cas des deux autres formats à gérer. De plus, la gestion des métadonnées se fait très simplement, en même temps que la création des tables tout comme l'ajout des clés primaires et des contraintes de domaines. Seules les clés étrangères sont ajoutées via la commande SQL « ALTER TABLE ». A la fin de la présentation des trois algorithmes, une figure indiquera les différences syntaxiques à ce sujet.

A noter que les attributs spatio-temporels se déclarent de la même manière qu'un attribut ordinaire. Cependant, nous doutons qu'ils soient gérés correctement et qu'il soit simple d'intégrer des données.

c) Script PostGreSQL / PostGIS

C'est ici qu'est utilisée la classe « PostGreSQL_PostGIS_Generator ».

Pour l'élaboration de ce script, j'ai travaillé avec le SGBD PGAdmin III. Celui-ci, grâce à son plugin « PostGIS », gère parfaitement les données spatio-temporelles. Ce format possède beaucoup plus de subtilités que le précédent, à savoir :

1. La gestion des types énumératifs : créer un type pour pouvoir l'utiliser par la suite
2. La gestion des contraintes de domaine : créer un domaine en posant les conditions souhaitées
3. La création des tables : la syntaxe varie selon que la table possède un attribut géométrique ou non
4. La gestion des métadonnées : à ajouter après la création des tables

Toutes ces subtilités rendent obligatoirement l'algorithme plus complexe, nous forçant à vérifier beaucoup plus d'informations et à gérer dans un ordre bien précis toutes les étapes à réaliser.

En soi, cela n'est pas difficile à implémenter, mais il a fallu effectuer de nombreuses recherches pour recueillir la globalité des solutions à mettre en place. Par exemple, la figure ci-dessous montre la différence entre la création d'une table qui possède un attribut géométrique ou non.

```
DROP TABLE IF EXISTS Commune;
CREATE TABLE Commune (
numInsee CHAR(5) NOT NULL,
nomCom VARCHAR(255) NOT NULL,
surfaceCom DOUBLE PRECISION NOT NULL,
PRIMARY KEY (numInsee ));
SELECT AddGeometryColumn('commune', 'geometriecom', 0, 'POLYGON', 2);

DROP TABLE IF EXISTS ParcelleConstructible;
CREATE TABLE ParcelleConstructible (
cos FLOAT NOT NULL,
numCadastre INT NOT NULL,
numInsee CHAR(5) NOT NULL,
PRIMARY KEY (numCadastre, numInsee ));
```

Figure 41 : Exemple de script PostGreSQL

Nous constatons que la table « Commune » possède un attribut géométrique nommé « geometriecom » de type « POLYGON ». Pour l'intégrer, il faut utiliser la fonction « AddGeometryColumn » fournie dans le plugin PostGIS prenant en paramètre le nom de la table qui recevra l'attribut géométrique, le nom de l'attribut géométrique, le SRID (référentiel géographique), le type de l'attribut et sa dimension.

La structure de la table pourrait encore être différente, en imaginant que la clé primaire soit l'attribut géométrique. En effet, cela serait impossible de préciser la clé primaire à la création de la table puisque l'attribut ne serait pas encore déclaré à ce moment-là, nous obligeant à spécifier la clé primaire plus tard via une commande « ALTER TABLE ».

d) Script MS ACCESS

Ce dernier format a été de loin le plus compliqué à générer. Bien évidemment, le SGBD utilisé était Microsoft ACCESS 2010. Contrairement aux deux SGBDs précédent, les données spatio-temporelles ne sont pas gérées.

La grande particularité de ce SGBD est qu'il ne permet pas d'exécuter un script SQL. Il autorise seulement d'effectuer une requête par une requête. Cela n'étant vraiment pas pratique, j'ai dû contourner le problème en générant un script sous un autre langage appelé Visual Basic. Microsoft ACCESS propose effectivement un éditeur Visual Basic. N'ayant que très peu utilisé ce langage, un gros travail de recherche était indispensable pour comprendre le mécanisme et surtout la syntaxe à adopter.

Cependant, l'algorithme mis en place dans la classe « MS_ACCESS_Generator » est assez similaire à celui de « PostGreSQL_PostGIS_Generator ». Les contraintes de domaine, la gestion des métadonnées et la gestion des types énumératifs étaient identiques. Ce qui changeait radicalement était lors de l'écriture dans le nouveau fichier.

La figure ci-dessous présente la création d'une table sous l'éditeur Visual Basic.

```
Sub DoSQL()

' Creation des tables

docmd.runsql "CREATE TABLE Riviere(" & _
"numRiviere INTEGER NOT NULL," & _
"nomRiviere VARCHAR(255) NOT NULL," & _
"longueur DOUBLE ," & _
"geometrieRiviere LINESTRING NOT NULL," & _
"CONSTRAINT PK_Riviere PRIMARY KEY (numRiviere ))"
```

Figure 42 : Création d'une table sous MS ACCESS

La syntaxe SQL en soi est conservée par la présence de « CREATE TABLE… » mais elle est imbriquée dans une fonction propre à Visual Basic nommée « docmd.runsql ». Contrairement à la fenêtre d'édition SQL, il est possible de mettre à la suite plusieurs fois la fonction « docmd.runsql », créant ainsi à la chaîne toutes les tables simultanément.

Tout le reste repose sur l'utilisation de fonctions de Visual Basic et de déclarations de variables. Le plus long et le plus difficile était donc de trouver ces fonctions et de les utiliser sans faire d'erreur de syntaxe. L'algorithme consiste toujours à créer le nouveau fichier, récupérer les tables dans le fichier XMI via une requête XPATH, etc…

Pour ne pas me répéter inutilement je vais donc simplement montrer à travers plusieurs figures les grandes différences syntaxiques entre les trois SGBDs.

AJOUT COMMENTAIRE EN POSTGRESQL

```
DROP TABLE IF EXISTS Commune;
CREATE TABLE Commune (
numInsee CHAR(5) NOT NULL,
nomCom VARCHAR(255) NOT NULL,
surfaceCom INT_UNSIGNED NOT NULL,
PRIMARY KEY (numInsee ));
COMMENT ON COLUMN Commune.surfaceCom
IS 'Commentaire de l attribut surfaceCom'
```

AJOUT COMMENTAIRE EN MYSQL

```
DROP TABLE IF EXISTS Commune;
CREATE TABLE Commune (
numInsee CHAR(5) NOT NULL,
nomCom VARCHAR(255) NOT NULL,
surfaceCom INT UNSIGNED COMMENT "Commentaire de l'attribut surfaceCom" NOT NULL,
geometrieCom POLYGON NOT NULL,
PRIMARY KEY (numInsee ));
```

AJOUT COMMENTAIRE SOUS MS ACCESS

```
Dim TD1 As DAO.TableDef
Dim ADesc1 As DAO.Property
Dim Att1 As DAO.Field

Set TD1 = DB.TableDefs("Commune")
Set Att1 = TD1.Fields("surfaceCom")
Set ADesc1 = Att1.CreateProperty("Description", dbText, "Commentaire de l'attribut surfaceCom")
Att1.Properties.Append ADesc1
```

Figure 43 : Ajout de commentaire sous les différents SGBDs

GESTION DE LA CONTRAINTE DE DOMAINE SOUS POSTGRESQL

```
CREATE DOMAIN INT_UNSIGNED
AS INT
CONSTRAINT INT_POSITIVE
CHECK (VALUE>=0);
```

GESTION DE LA CONTRAINTE DE DOMAINE SOUS MYSQL

```
DROP TABLE IF EXISTS Commune;
CREATE TABLE Commune (
numInsee CHAR(5) NOT NULL,
nomCom VARCHAR(255) NOT NULL,
surfaceCom INT UNSIGNED,
geometrieCom POLYGON NOT NULL,
PRIMARY KEY (numInsee ));
```

GESTION DE LA CONTRAINTE DE DOMAINE SOUS MS ACCESS

```
Dim TDef1 As DAO.TableDef
Dim Attr1 As DAO.Field

Set TDef1 = DB.TableDefs("Commune")
Set Attr1 = TDef1.Fields("surfaceCom")
Attr1.ValidationRule = ">0"
Attr1.ValidationText = " Entrer un nombre supérieur a 0"
```

Figure 44 : Gestion de la contrainte de domaine sous les 3 SGBDs

D'autres différences surviennent lors de la création d'un type énumératif par exemple. Le but est de montrer à quel point les scripts générés proposent une syntaxe complètement différente pour représenter strictement la même structure.

A la fin de l'exécution du second parseur, nous avons donc en possession des scripts prêts à l'emploi, que ce soit pour MS ACCESS, MySQL ou PostGreSQL. Ils sont stockés dans le dossier « Fichiers_MRD_Générés ».

Cependant, pour être sûr de la fiabilité des scripts fournis en sortie, de multiples tests ont dû être réalisés. De plus, nous avons pu tester le programme sur un cas concret proposé par Mme Nathalie Saint-Geours.

V. Tests

De nombreux Modèles Conceptuels de Données (MCD) ont été produits tout au long de l'implémentation des parseurs, afin de vérifier leur fiabilité. Cette phase a débuté dès lors que le premier parseur était capable de fournir des MRD. Tout projet nécessite cette étape indispensable pour valider ou non le travail fourni.

1. Tests unitaires

a) Tests pour le parseur MCD to MRD

Les premiers tests unitaires étaient basés sur des MCD très simplistes comprenant deux ou trois classes avec peu d'attributs et des associations binaires ordinaires (pas d'héritage, de composition, d'association réflexive…). De cette façon j'ai pu mettre en place les transformations standards basiques en changeant seulement les multiplicités pour observer le comportement du parseur. La démarche est itérative et prend du temps à savoir : créer dans un premier temps notre MCD sous BOUML puis l'exporter au format XMI et enfin lancer le parseur « MCDtoMRD ». Tout ce processus est à répéter même pour des modifications infimes telles que :

- Modification d'une multiplicité
- Modification d'un type pour un attribut
- Ajout du stéréotype <<id>> ou <<nullable>>
- Ajout d'une classe, d'un attribut…

Au fur et à mesure, j'ai donc ajouté de nouveaux éléments tels que les classes d'associations, les liens de compositions, d'héritages, etc… Je pouvais donc me concentrer sur une partie précise de l'algorithme et ajouter brique après brique. Il n'y a évidemment que les tests pour nous rendre compte des bugs ou des erreurs à corriger. Pour en revenir à un problème cité précédemment, c'est d'ailleurs grâce à des MCD fournis par mon tuteur Michel Passouant que bon nombre de ces erreurs ont été détectées. Par exemple, gérer les liens de composition et d'héritage comme je l'ai expliqué dans la partie algorithmique. Un autre MCD proposé par mon tuteur a mis en évidence un problème auquel je n'avais pas pensé, présenté ci-dessous.

Nous voyons là tout l'intérêt d'une phase de test avec des cas pilotes.

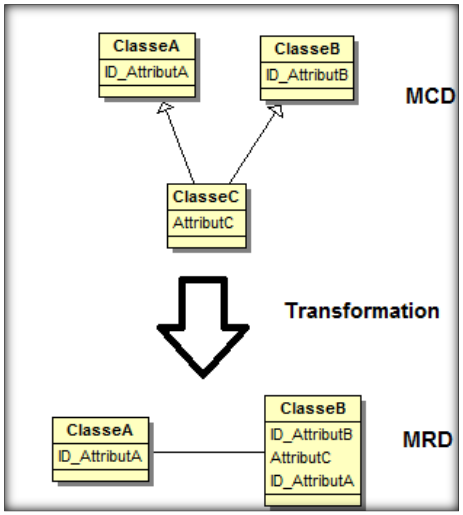

Figure 45 : Exemple d'héritage multiple

Il s'agit de l'héritage multiple. Ce cas est totalement anodin si lors des options proposées par l'interface graphique l'utilisateur a choisi de conserver toutes les classes en dupliquant la clé primaire des classes mères dans les classes filles. Cependant, s'il choisit de supprimer les classes filles en faisant tout remonter dans les classes mères, cela devient beaucoup délicat. Nous pouvons constater que « ClasseC » hérite de « ClasseA » et « ClasseB ». Dans ce cas précis, comment savoir si « AttributC » doit être transféré dans « ClasseA » ou « ClasseB » ? Car le transférer dans les deux classes provoquerait une redondance d'information. C'est pourquoi ce cas fait partie des rares cas pour lesquels l'utilisateur est sollicité, comme le montre la figure ci-dessous.

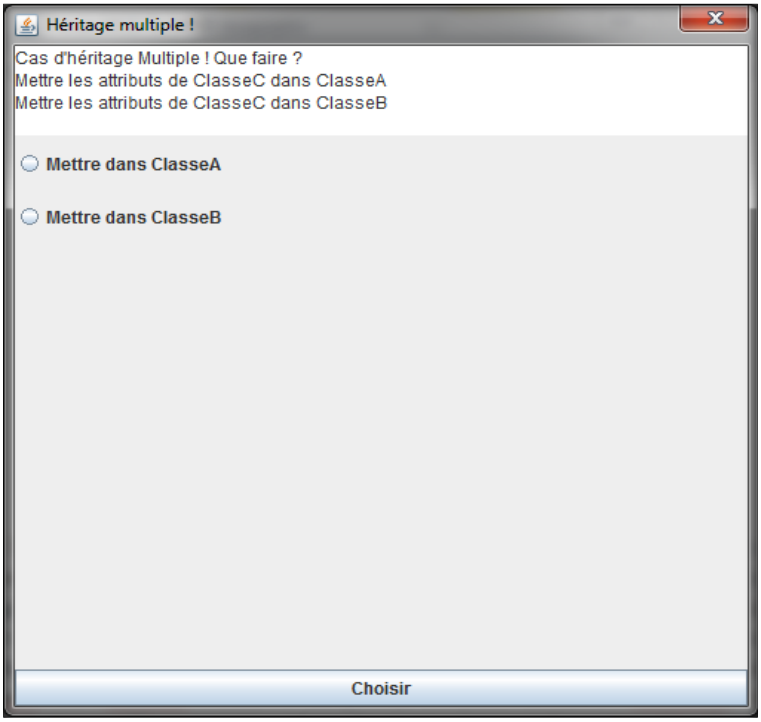

Figure 46 : Interface de l'héritage multiple

Mais cela va encore plus loin que de simplement choisir la classe dans laquelle faire migrer les attributs de « ClasseC ». Les étapes suivantes sont à réaliser :

1. Déplacer les attributs de « ClasseC » dans la classe désignée par l'utilisateur
2. Supprimer « ClasseC »
3. S'ils le ne le possédaient pas, rajouter à tous les attributs venant de « ClasseC » le stéréotype <<nullable>>
4. Ajouter les identifiants des autres classes mères dans celle désignée par l'utilisateur en tant que clés étrangères (dans l'exemple, « ID_AttributA » devient clé étrangère dans « ClasseB »)

La phase de tests a montré qu'il fallait étudier rigoureusement tous les cas possibles sous peine de créer des MRD erronés. Ce qui rend la tâche encore plus délicate est que ces erreurs ne sont pas détectables par le parseur. En effet, il est possible de lever des exceptions lorsque le fichier XMI fourni en entrée possède une structure qui ne correspond pas au XSD, mais déceler une erreur liée à la sémantique n'est possible que par l'expertise d'un humain. C'est pourquoi le nombre de MCD créé a dépassé sans aucun doute la centaine, en faisant systématiquement une double vérification :

- Exécution des parseurs
- Analyse en détail des résultats par rapport à des résultats obtenus manuellement

De plus, continuer avec un MRD erroné entraîne inévitablement un script SQL comportant des erreurs, même si celles-ci n'empêchent pas d'importer le script dans le SGBD, la sémantique et la structure ne seront pas correctes. Ceci nous amène donc au fait qu'il ne suffisait pas de vérifier le passage du MCD au MRD, ceci ne correspondant qu'à 50% du chemin accompli.

b) Tests pour le parseur MRD to SQL

Réussir à importer le script dans le SGBD ne signifie pas pour autant que le résultat obtenu est celui attendu par l'utilisateur. Cependant, réussir cette étape est déjà un point essentiel, signifiant qu'aucune erreur syntaxique n'est présente. Mais comme précédemment, les erreurs de sémantique peuvent provoquer des erreurs, comme l'atteste l'exemple suivant.

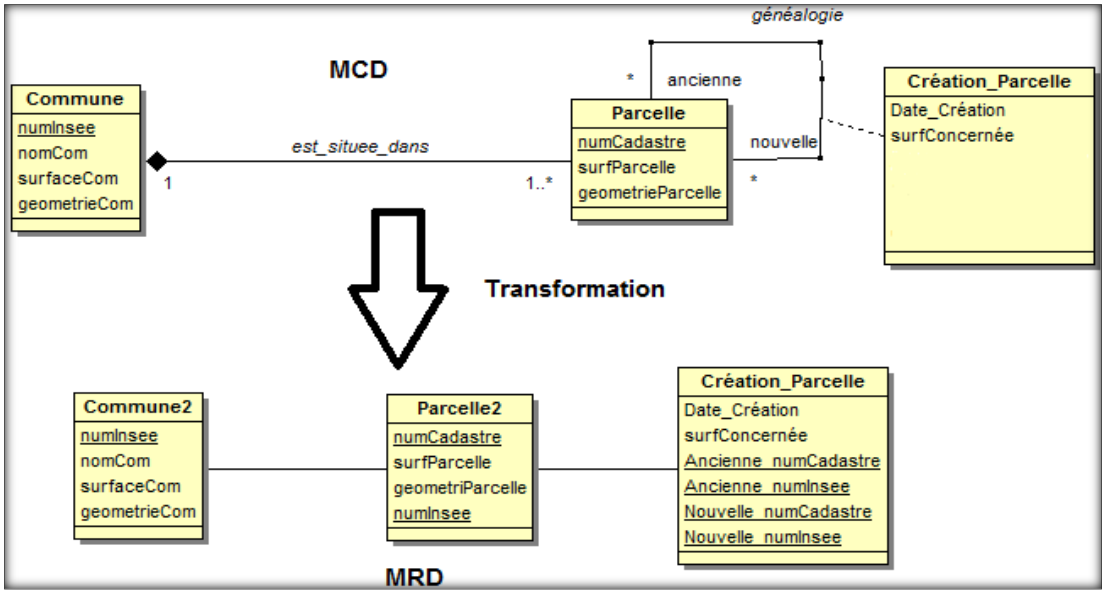

Figure 47 : Cas spécial de réflexivité

L'exécution du programme « MCD to MRD » produit le MRD correct. Seulement, lors de l'exécution du script dans PGAdmin III, un message d'erreur indiquait qu'il était impossible d'ajouter une contrainte de clé étrangère entre la table « Parcelle » et « Création_Parcelle ». En analysant le script généré, nous nous sommes aperçus que mon parseur écrivait un résultat incorrect d'un point de vue sémantique. En effet, je ne créais qu'une seule clé étrangère de cette façon :

```
ALTER TABLE Création_Parcelle
ADD CONSTRAINT fk_Parcelle_Création_Parcelle
FOREIGN KEY ( ancienne_numCadastre,ancienne_numInsee, nouvelle_numCadastre,nouvelle_numInsee)
REFERENCES Parcelle (numCadastre,numInsee);
```

DEFINITION DE CLE ETRANGERE INCORRECTE

```
ALTER TABLE Création_Parcelle
ADD CONSTRAINT fk_Parcelle_Création_Parcelle1
FOREIGN KEY ( ancienne_numCadastre,ancienne_numInsee)
REFERENCES Parcelle (numCadastre,numInsee);

ALTER TABLE Création_Parcelle
ADD CONSTRAINT fk_Parcelle_Création_Parcelle2
FOREIGN KEY ( nouvelle_numCadastre,nouvelle_numInsee)
REFERENCES Parcelle (numCadastre,numInsee);
```

DEFINITION DE CLE ETRANGERE CORRECTE

Figure 48 : Création de clés étrangères

L'erreur venait donc du fait que sémantiquement, lors de la définition d'une clé étrangère, le nombre d'attributs spécifiés à la ligne « FOREIGN KEY » doit être identique à celui présent à la ligne « REFERENCES ». Or, je voulais que 4 attributs deviennent clés étrangères de seulement 2 attributs.

La solution présentée est donc de déclarer deux clés étrangères en spécifiant dans la ligne « REFERENCES » strictement les mêmes informations.

Voici donc les principales anomalies détectées lors des tests. D'autres erreurs ont été corrigées mais il est inutile de toutes les citer. La phase de tests a duré presque aussi

longtemps que la phase d'implémentation, car j'ai rectifié la majorité des erreurs au début mais je n'ai repéré les plus subtiles qu'en milieu voire en fin de stage.

Par ailleurs, la spécificité de certains cas limites discutés dans la section suivante demandera un développement ultérieur qui sera réalisé dans les futurs travaux de l'équipe.

c) Cas limites

i. Cas spécial lors de l'héritage

Le premier cas limite auquel nous sommes confrontés est celui présenté ci-dessous.

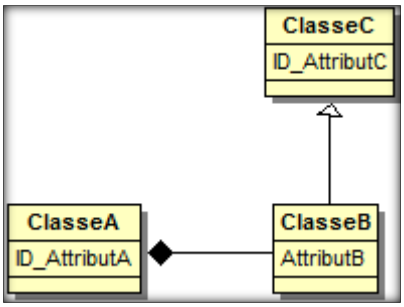

Figure 49 : Premier cas limite

Ce cas limite se présente seulement si l'utilisateur a choisi comme option de supprimer les classes filles et de faire remonter ses attributs dans la classe mère. En effet, dans ce cas, dans quel ordre exécuter la composition et l'héritage ?

- Composition puis héritage : « ID_AttributA » est transféré dans « ClasseB » en tant que clé étrangère et clé primaire puis « ClasseB » disparaît après avoir fait remonter « AttributB » et « ID_AttributA » dans « ClasseC ». « ID_AttributA » ne peut plus être clé primaire dans « ClasseC » car cela provoquerait une erreur sémantique mais alors on perdrait la notion de clé primaire…

- Héritage puis composition : « AttributB » est transféré dans « ClasseC » puis « ClasseB » disparaît. Que devient alors la composition ? « ID_AttributA » doit-il être transféré en tant que clé étrangère et clé primaire dans « ClasseC » ?

ii. Date en tant que clé primaire

Le second cas limite concerne la date. Dans certains cas, la date doit être intégrée à la clé primaire de certaines relations. Ceci permet de stocker plusieurs tuples caractérisant des mêmes informations ayant lieu à des dates différentes. D'un point de vue modélisation, il est alors nécessaire de représenter la date comme une entité à part entière.

Malheureusement, ceci nécessite de pouvoir ajouter des associations n-aires, élément qui n'est pas géré par BOUML.

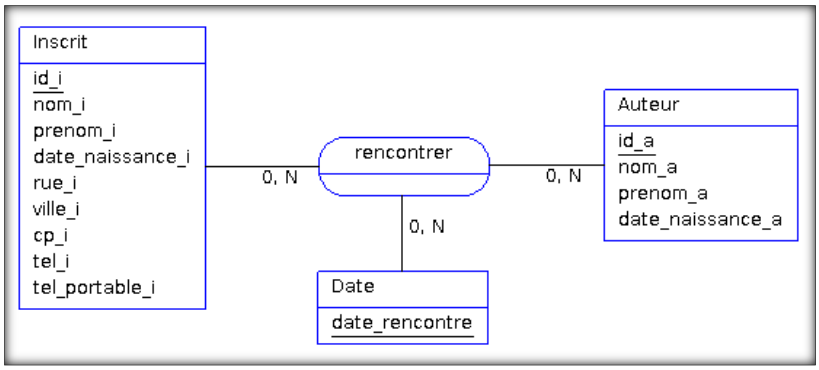

Figure 50 : Association ternaire impliquant la date

Cette figure représente le problème précisé ci-dessus. Nous avons à présent listé une très grande partie des tests unitaires réalisée durant le stage. Un autre élément a été très instructif et très efficace pour mettre en application le projet. Il s'agit du cas d'étude réalisé avec Mme Nathalie Saint-Geours.

2. Evaluation globale sur un cas d'étude: Analyse de sensibilité de modèles spatialisés : Application à l'analyse coût-bénéfice de projets de prévention des inondations

Le 15 mai, soit 1 mois et demi après le début de mon stage, j'ai participé au « Stagiairothon » organisé par le laboratoire TETIS. Durant cette journée, nous étions une dizaine de stagiaires à présenter notre projet. Quelques jours plus tard, Nathalie Saint-Geours ayant assisté à la manifestation m'a contacté en précisant que le sujet l'intéressait et qu'elle me proposait de travailler sur les résultats de sa thèse en télédétection, me servant donc de cas d'étude. Mes tuteurs et moi avons trouvé l'idée très intéressante, raison pour laquelle nous avions organisé à la mi-juin une réunion en présence de Mathieu Roche, Michel Passouant, Sophie Fortuno, Nathalie Saint-Geours et moi-même.

a) Le contexte

La réunion s'est déroulée sur deux heures pendant lesquelles Nathalie Saint-Geours nous a expliqué en détail sur quoi reposait le sujet de sa thèse. Durant trois ans, elle a eu pour terrain d'étude le bassin versant de l'Orb. L'Orb est un fleuve côtier situé dans l'Hérault se jetant dans la mer Méditerranée. La plus grande commune traversée est Béziers. Un objectif de sa thèse était d'établir une analyse de coût-bénéfice de prévention du risque d'inondation de cette région. Cette analyse était nécessaire du fait que le bassin versant de l'Orb subit fréquemment des crues et peut donc causer potentiellement des dégâts sur les territoires environnants. Les coûts variant

énormément en fonction de l'occupation du sol, ces derniers ont été divisés en quatre grandes catégories :

- Les habitations
- Les cultures
- Les entreprises
- Les campings

En fonction de nombreux paramètres Nathalie Saint-Geours a pu établir le coût des dégâts causés pour chacune de ces catégories.

La figure 51 représente son terrain d'étude.

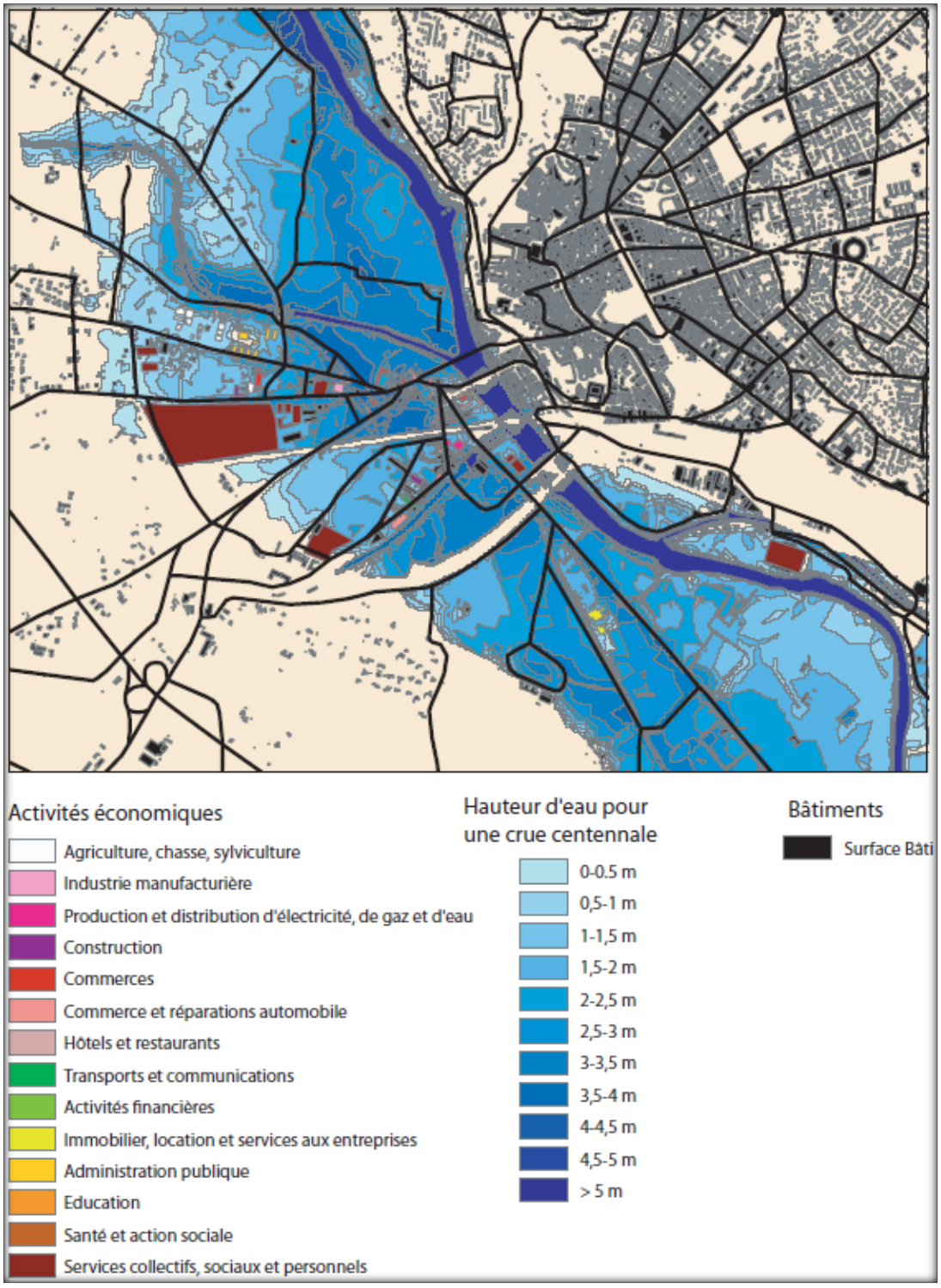

Figure 51 : Terrain d'étude

Durant sa thèse, Nathalie Saint-Geours n'a pas utilisé de SGBD et a donc stocké ses données dans des fichiers séparés sans réflexion particulière sur leur structuration. C'est la raison pour laquelle elle souhaitait que nous reprenions ensemble ses données, en construisant un Modèle Conceptuel de Données pour créer au final une base de données sous PGAdmin III.

b) Les données initiales

Lors de sa thèse, les données à la disposition de Nathalie Saint-Geours n'étaient pas centralisées. Les données à caractéristiques géographiques ont été récupérées par plusieurs moyens :

- Par la chambre du commerce
- Données de l'agriculture 2009
- L'IGS
- Données sur le terrain obtenues par des stagiaires

Ces données géographiques ont été regroupées sous un projet QGIS (terrain d'étude, emplacement des bâtiments, habitations, campings, cultures, hauteur d'eau, etc…) mis à disposition pour l'évaluation de mon projet.

Les autres données ont été calculées à l'aide de scripts établis par Nathalie Saint-Geours. Il s'agit des coûts monétaires engendrés par type de bâtiments en fonction de plusieurs paramètres tels que :

- La hauteur d'eau
- Le type de bâtiment
- La probabilité de crues par zone (décennale, centennale, millénale)

Ces éléments sont stockés dans quatre fichiers CSV, chacun correspondant à un type de bâtiment.

Durant la réunion, nous avons pu donc établir un MCD à l'aide de toutes ces informations, aboutissant à la figure 52.

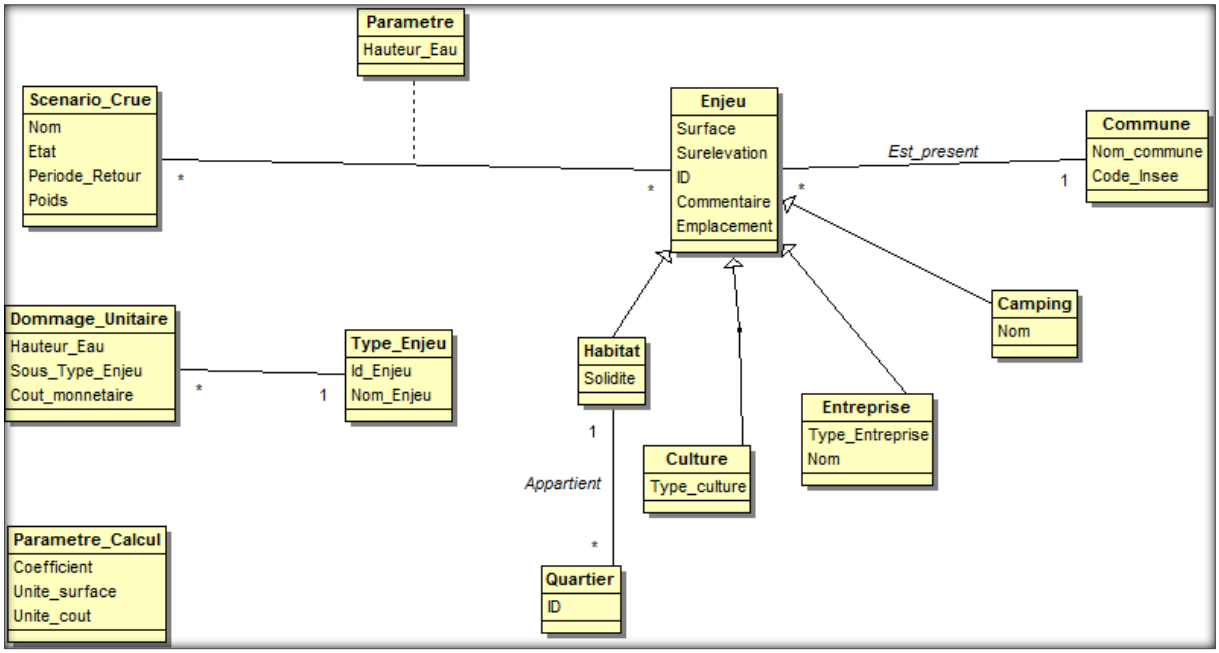

Figure 52 : MCD correspondant au cas d'étude

c) Implémentation de la base et intégration des données

A l'issue de la réunion j'ai donc généré le script au format PostGreSQL pour l'importer dans PGAdmin III. C'est ici que nous voyons réellement la force et l'utilité de l'outil que nous implémentons : grâce à lui nous avons pu nous concentrer intégralement sur le MCD sans nous préoccuper un seul instant des transformations qu'il faudrait réaliser plus tard pour le MRD.

Cependant, intégrer les données dans la base demeurait le plus compliqué. J'ai fait un travail de restructuration de l'ensemble des données pour qu'elles soient importables dans les différentes tables. Les CSV ont été modifiés afin de venir alimenter les tables « Habitat », « Culture », « Entreprise » et « Camping ». Les données géographiques présentes sous QGIS ont été exportées également au format CSV puis ont subis des modifications dans leur structure pour venir alimenter le reste des tables.

Exploiter les données sous QGIS m'a demandé un effort supplémentaire car j'ai dû me familiariser avec ce logiciel que je n'avais jamais utilisé. C'est pourquoi Michel Passouant m'a aidé durant une après-midi pour effectuer des manipulations sous QGIS afin de calculer des données qui m'étaient utiles. J'ai notamment réalisé avec son aide des statistiques de zone en croisant les données d'une couche raster (image matricielle ou « carte de points », constituée d'une matrice de points colorés) avec une couche vectorielle (image numérique composée d'objets géométriques individuels que nous pouvons étudier à différentes échelles sans perte de qualité). La couche raster contenait les informations concernant les hauteurs d'eau pour le terrain d'étude. De son côté, la couche vectorielle contenait les informations des différents bâtiments. En croisant ces deux couches, nous en avons obtenu une nouvelle contenant les informations des bâtiments en relation avec leurs hauteurs d'eau associées.

Alimenter la base de données a été incontestablement l'opération la plus longue et la plus difficile. Malgré cela, j'ai réussi à présenter à Nathalie Saint-Geours nos résultats.

d) Résultats proposés

Une fois la base opérationnelle, il ne me restait plus qu'à exploiter la nouvelle structure de données sous QGIS. Pour cela j'ai connecté la base présente sous PGAdmin III avec QGIS. Une fois la connexion établie, QGIS permet de visualiser les tables possédant des attributs géométriques en l'ajoutant en tant que couche. De base, seule la table « Enjeu » est donc concernée puisqu'elle dispose de l'attribut « Emplacement ». Cet attribut est de type « POLYGON » et contient les données géographiques de l'ensemble des bâtiments.

Ceci est seulement le début puisqu'à ce stade là nous disposons des mêmes informations qu'avait Nathalie Saint-Geours. Ce qui va démontrer la puissance de notre solution est la méthode pour obtenir les coûts générés pour chaque type de bâtiment.

Pour rappel, Nathalie Saint-Geours calculait ces coûts via des scripts développés complexes appliqués sur des CSV. Cette méthode nécessitait donc un temps relativement important pour écrire ces scripts. De plus, les résultats étaient observables uniquement dans des fichiers CSV.

Maintenant, tous les coûts sont obtenus à l'aide de simples requêtes sur les différentes tables. Le principe est simple : créer une vue contenant l'emplacement du bâtiment concerné, son identifiant et plus spécialement son coût monétaire en fonction de sa hauteur d'eau. Toutes ces informations sont disponibles en effectuant de simples jointures.

Prenons l'exemple suivant : obtenir les coûts monétaires des cultures en fonction de la céréale cultivée et de sa hauteur d'eau. La requête correspondante est la suivante.

```
create or replace view Couts_Cultures as
SELECT distinct C.type_culture, C.newpk_fk_id, E.emplacement, D.cout_monnetaire, P.hauteur_eau
FROM Culture C, Enjeu E, Dommage_unitaire D, Parametre P
WHERE C.newpk_fk_id = E.id AND C.newpk_fk_id = P.new_pk_fk_id AND round(CAST(P.hauteur_eau as numeric), 2) = D.hauteur_eau
AND C.type_culture = lower(D.sous_type_enjeu);
```

Figure 53 : Exemple de requête sur les cultures

Nous venons donc d'obtenir en une seule requête les mêmes résultats que Nathalie Saint-Geours récupérait à l'aide d'un script beaucoup plus complexe et long à mettre en place.

En plus du coût monétaire qui est l'information que nous recherchons, conserver l'emplacement de l'élément est primordial. Comme je le disais précédemment, QGIS permet de visualiser les tables possédant un attribut géométrique. Ce qui est fort pratique est que cela s'applique également aux vues. Donc le fait de créer une vue avec les informations qui nous intéressent combinées avec les données géographiques va nous permettre de l'ajouter en tant que couche géographique visualisable.

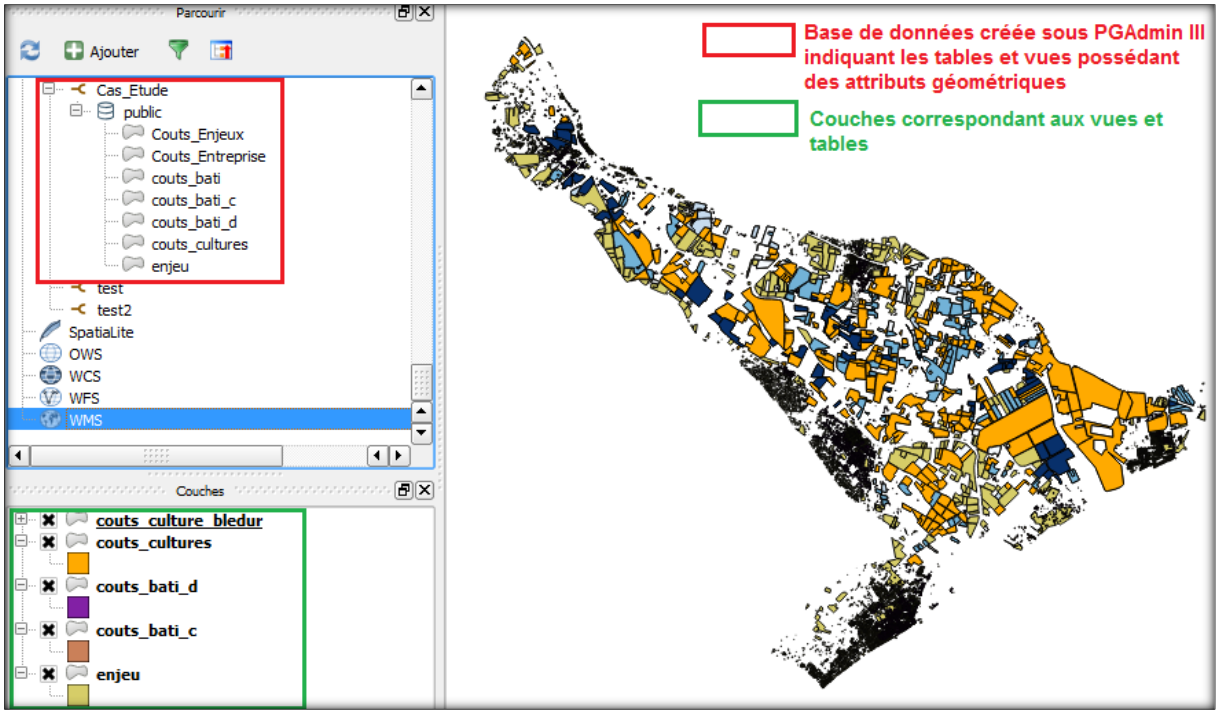

Figure 54 : Résultats sous QGIS

La figure 54 confirme ce que nous venons de dire. La vue « Couts_Cultures » est présente dans la base de données « Cas_Etude », et également dans l'onglet « Couches ». Elle est représentée en orange sur la carte. Les autres couleurs visibles correspondent aux autres vues ajoutées. Le dégradé de couleur bleue est une option proposée par QGIS très utile. Cette technique s'applique ci-dessus à la vue « Couts_Culture_bledur », qui comme son nom l'indique est le résultat d'une requête ne concernant que les cultures de blé dur. Le dégradé apporte une notion sémantique forte et visuelle : toutes les cultures de blé dur représentées en bleu foncé correspondent à des zones fortement inondées donc ayant un coût monétaire important. Plus le bleu est clair, moins la zone est touchée.

L'autre force de combiner la base de données avec QGIS est représentée par la figure suivante.

Figure 55 : Table d'attributs sous QGIS

Nous voyons qu'une parcelle est surlignée en rouge. Sa couleur de base est bleu foncé. Ceci signifie que cette parcelle est une culture de blé dur. En ouvrant la table d'attributs, nommée ici « Attributs d'entités », nous avons accès à toutes les informations contenues dans la vue « Couts_Culture_bledur » pour cet élément précis. Ceci équivaut à une requête « SELECT FROM WHERE ». Il y a donc plusieurs moyens pour consulter les informations en notre possession, l'une entièrement technique par le biais de requêtes, et l'autre visuelle via l'utilisation des couches sous QGIS. Pouvoir manipuler les données sur une carte permet de les interpréter différemment et plus simplement, car un support visuel est toujours plus parlant qu'une gigantesque table recensant des milliers de lignes de données.

Nous avions organisé une seconde réunion avec Nathalie Saint-Geours et mes tuteurs fin juillet durant laquelle j'ai exposé les méthodes utilisées et les résultats présentés ci-dessus. Nous avons tous été très satisfaits des résultats obtenus et d'accords pour dire que structurer les données dès le départ simplifie énormément la suite des opérations. De plus, cette modélisation apporte une fiabilité et une intégrité des données que Nathalie Saint-Geours ne possédait pas avec son ancienne implémentation. En effet, la structure créée par le biais d'un MCD lie toutes les données et supprime la redondance d'information. Les coûts monétaires sont désormais gérés de la même manière que les données géographiques dans une base de données, contrairement à la méthode initiale où ces données étaient stockées de manière indépendante dans des fichiers CSV séparés. Ce cas d'étude a également permis de vérifier le bon fonctionnement des différents outils implémentés.

L’annexe technique fournie contient plus de détails concernant le cas d’étude.

L’intégralité de ces tests ont permis de modifier, corriger et affiner tout au long du projet les deux parseurs pour au final les valider et les rendre fonctionnels.

VI. Discussion

Tout au long du projet, quelques écueils sont venus freiner l'avancée du stage. Heureusement, aucun d'entre eux n'a réellement été important car en général ceux-ci étaient réglés en quelques jours maximum. Par exemple, lors de la mise en place du XSD, les différents espaces de nom « :xmi » et « :uml » étaient compliqués à gérer. Je l'ai d'ailleurs déjà mentionné précédemment, car même si ce dernier a été corrigé rapidement, il n'en demeurait pas moins un problème inévitable à régler.

Ensuite, l'implémentation des parseurs s'est déroulée sans véritable soucis. Cependant, ce qui a rendu leur élaboration peu évidente a été le nombre très important de transformations à prendre en compte. Régulièrement, nous faisions le point avec mes tuteurs concernant les différents cas de figures. Certains ont prêté à débat, tels que les liens de spécialisation et de généralisation. Le « problème » récurrent était la découverte de nouveaux cas spéciaux entraînant des modifications dans les algorithmes des parseurs. Celui que nous pouvons noter a été présenté et concernait la gestion entre les liens de spécialisation et les liens de composition. Ceci m'a obligé à transformer de manière importante des briques que j'avais implémentées au début du travail de stage.

Le nombre important d'outils à utiliser était également un facteur non négligeable. J'ai effectivement dû jongler entre trois SGBDs et appréhender des logiciels que je n'avais jamais utilisés tels que QGIS ou PowerAMC. La prise en main et la configuration de chacun d'entre eux nécessite du temps et de faire de nombreuses recherches pour connaître en détail tous les aspects qui nous sont utiles.

Enfin, pour en revenir au début du projet, l'analyse du format XMI a été longue et fastidieuse. La raison est que XMI est censé être un standard alors qu'il n'en est pas vraiment un. Le problème est qu'il y a autant de structures XMI différentes que de logiciels UML présents sur le marché, bien qu'ils utilisent les mêmes versions de XMI. Analyser un tel format est donc relativement compliqué du fait de son hétérogénéité. De plus, XMI ne possède pas de documentation très claire et rend encore plus difficile sa compréhension. J'ai donc dû éclaircir avec le peu d'informations disponibles cette structure et en comprendre la logique. Ceci m'a permis de développer une « recherche fouillé» et d'adopter in fine une démarche itérative pour conduire mon projet.

Grâce à un travail et des réunions régulières, une programmation et un suivi du projet, les objectifs initiaux ont été atteints. En témoigne le diagramme de Gantt présenté ci-dessous.

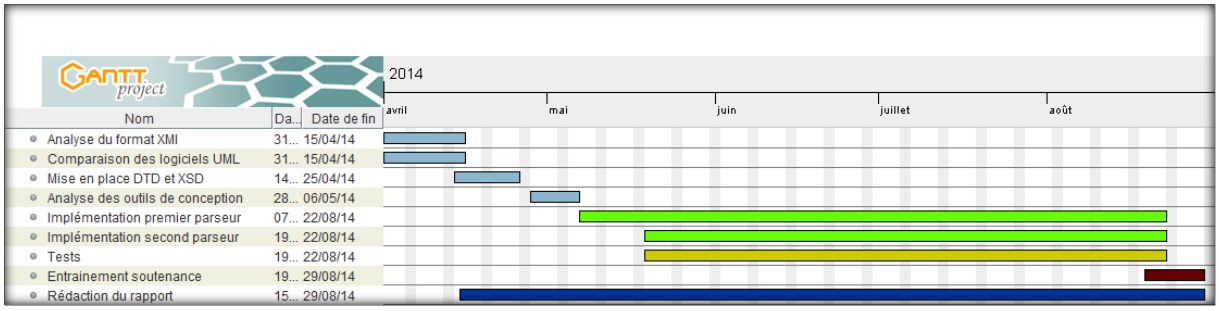

Figure 56 : Diagramme de Gantt

Ce stage se concrétise par un développement de mes connaissances et par une expérience acquise en méthodologie et en recherche d'informations appliquées à un projet.

VII. Bilan et perspectives

Nous arrivons au terme de ce rapport. Sur le plan personnel, je tire un bilan très positif sur l'ensemble du stage. J'ai approfondi mes connaissances sur le langage JAVA en utilisant de manière avancée la librairie JDOM. La notion de « parseur » est maintenant maîtrisée. J'ai également développé mon esprit d'analyse et de recherches en étudiant dans le détail le format XMI. Enormément de nouvelles connaissances se sont ajoutées à mon champ de compétences telles que l'élaboration d'un XSD, l'utilisation des logiciels de modélisation UML (plus particulièrement BOUML…), la manipulation de QGIS et PowerAMC ou encore la création d'un installeur pour faciliter la distribution d'un logiciel.

De plus, j'ai eu la chance de bénéficier d'un encadrement très rapproché de mes tuteurs Michel Passouant, Mathieu Roche et Sophie Fortuno qui ont parfaitement joué leur rôle en étant présent durant la totalité du stage et en me guidant semaine après semaine. Ils m'ont incité à fournir un travail rigoureux, ce rapport en faisant partie. Je n'oublie pas le « Stagiairothon » que j'ai déjà cité précédemment. Ce dernier a été une expérience très enrichissante car il m'a forcé à présenter mon sujet de stage de la manière la plus compréhensible possible pour des personnes ayant des compétences dans des domaines totalement différents. Cela a également été bénéfique pour moi, m'offrant une nouvelle vision de mon propre travail.

Concernant les outils implémentés, mes tuteurs et moi sommes extrêmement satisfaits du résultat obtenu. Toutefois, quelques fonctionnalités supplémentaires pourraient être ajoutées comme par exemple essayer de gérer les associations n-aires en modifiant le code source de BOUML. Ceci règlera le cas limite évoqué concernant la date.

En somme, ce stage constitue pour moi une expérience très importante sur le plan humain et pour mon avenir professionnel. Je tiens à remercier une dernière fois toutes les personnes que j'ai pu côtoyer au sein du CIRAD.

En espérant que le travail effectué servira au grand projet qu'est le « Patrimoine Numérique ». Il apporte une solution opérationnelle et accessible pour répondre à la problématique de structuration et de description des jeux de données. Cet outil est un appui pour aider les chercheurs à bien définir et collecter les données nécessaires à leurs recherches, et aussi faciliter la ré-utilisation future de ces données.

VIII. Bibliographie

[1] http://www.omg.org

[2] http://books.google.fr/books?id=WN60A9kr-kUC

[3] http://fr.wikipedia.org

[4] http://www.developpez.net

[5] http://stoyanov.org/books/Mastering%20XMI%20-%20Java%20Programming%20with%20XMI,%20XML,%20and%20UML%20%28Wiley%29.pdf

[6] http://www.jdom.org/docs/apidocs/

[7] http://postgis.net/docs/manual-1.3/

[8] http://forums.postgresql.fr/

[9] http://cuiwww.unige.ch/~guyot/TPBD1/BD/UML-BDR.pdf

[10] http://www.irit.fr/~Thierry.Millan/CNAM-NFP107/UML%20et%20les%20Bases%20de%20Donn%C3%A9es.pdf

[11] http://www.deg.byu.edu/papers/DataMappings.pdf